[스파르타코딩클럽] 프론트엔드의 꽃, 리액트 - 3주차

매 주차 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/88f15680-77f6-4acc-a94e-b4693d9eae05/___3_.pdf

[수업 목표]

1. React-router로 주소에 따라 다른 페이지를 보여줄 수 있다.
2. 미리 정해놓은 주소가 아닐 때, '앗! 잘못 찾아오셨어요!' 페이지를 보여줄 수 있다.
3. Redux로 상태관리를 해보고 아래의 상태관리 흐름을 이해한다.
   (기본 값이 있고 → 어떤 동작을 하면("어떤 동작을 하는 지 정하자! → 하면 뭐가 바뀌는 지 정하자!" 과정이 사전에 필요하다!) → 값이 변했잖아? → 컴포넌트한테 알려주자!)
4. Redux hook을 사용해본다.

[목차]

모든 토글을 열고 닫는 단축키
Windows : `Ctrl` + `alt` + `t`
Mac : `⌘` + `⌥` + `t`

01. 라우팅이란

▼ 1) SPA란?

> 👉 **Single Page Application!**
> 말 그대로 서버에서 주는 html이 1개 뿐인 어플리케이션이에요.
> 전통적인 웹사이트는 페이지를 이동할 때마다 서버에서 html, css, js(=정적자원들)을 내려준다면, SPA는 딱 한번만 정적자원을 받아옵니다.

- 왜 굳이 html을 하나만 줄까?

  → 많은 이유가 있지만, 그 중 제일 중요한 건 **사용성** 때문입니다.

  페이지를 이동할 때마다 서버에서 주는 html로 화면을 바꾸다보면 상태 유지가 어렵고, 바뀌지 않은 부분까지 새로 불러오니까 비효율적이거든요.
  (사용자가 회원가입하다가 적었던 내용이 날아갈 수도 있고,
  블로그같은 경우, 페이지마다 새로 html을 받아오면 바뀐 건 글 뿐인데 헤더와 카테고리까지 전부 다시 불러와야 합니다.)

- 단점은 없나?

  → 단점도 있어요. SPA는 딱 한 번 정적자원을 내려받다보니, 처음에 모든 컴포넌트를 받아옵니다.

  즉, 사용자가 안들어가 볼 페이지까지 전부 가지고 옵니다. 게다가 한 번에 전부 가지고 오니까 아주아주 많은 컴포넌트가 있다면 첫 로딩 속도가 느려집니다.

▼ 2) 라우팅이란?

> 👉 SPA는 주소를 어떻게 옮길 수 있을까?
> html은 딱 하나를 가지고 있지만, SPA도 브라우저 주소창대로 다른 페이지를 보여줄 수 있어요.
> 이렇게 브라우저 주소에 따라 다른 페이지를 보여주는 걸 라우팅이라고 부릅니다.

- 전부 직접 구현하나요?

  → 이미 만들어진 라우팅 라이브러리가 있습니다!

  우리는 리액트 사용자들이 가장 많이 쓰는 라우팅 라이브러리를 가져와서 사용해볼거예요.

02. 리액트에서 라우팅 처리하기 (1)

▼ 4) react-route-dom 패키지 설치하기

- 먼저 새프로젝트를 만들어주세요!

  > 👉 2주차 강의에서 새 프로젝트를 만들 때 해야하는 것 STEP 1~3을 알려드렸던 거 기억하시죠!
  > 기억이 안나신다면 2주차 강의자료 7번을 보고 오세요!😉

- react-route-dom 설치

```
yarn add react-router-dom
```

▼ 5) react-route-dom 공식 문서를 보자!

> 👉 react-router-dom 공식 문서를 보면서 하나씩 따라서 설정해볼게요. 🙂
> 공식문서는 대부분 비슷하게 생겼으니까, 함께 공식문서를 따라하면서 문서 보는 법 익혀봅시다!
> **공식문서 보러가기→**

03. 리액트에서 라우팅 처리하기 (2)

▼ 6) 페이지를 전환해보자!

▼ (1) index.js에 BrowserRouter 적용하기

- BrowserRouter 적용하기

```
import React from 'react';
import ReactDOM from "react-dom";
import { BrowserRouter } from "react-router-dom";
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

// 이부분이 index.html에 있는 div#root에 우리가 만든 컴포넌트를 실제로 랜더링하도록 연결해주는 부분입니다.
ReactDOM.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
  document.getElementById("root")
);

// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

> 👉 BrowserRouter(브라우저라우터)는 웹 브라우저가 가지고 있는 주소 관련 정보를 props로 넘겨주는 친구입니다.
> 현재 내가 어느 주소를 보고 있는 지 쉽게 알 수 있게 도와줘요.

▼ (2) 세부 화면 만들기

- Home.js

```
import React from "react";

const Home = (props) => {

    return (
        <div>메인 화면이에요.</div>
    )
}

export default Home;
```

- Cat.js

```
import React from "react";
```

```
const Cat = (props) => {

    return (
        <div>고양이 화면이에요.</div>
    )
}

export default Cat;
```

- Dog.js

```
import React from "react";

const Dog = (props) => {

    return (
        <div>강아지 화면이에요.</div>
    )
}

export default Dog;
```

▼ (3) App.js에서 Route 적용하기

- Route 사용방법 1: 넘겨줄 props가 없을 때

```
<Route path="주소[/home 처럼 /와 주소를 적어요]" component={[보여줄 컴포넌트]}/>
```

- Route 사용방법 2: 넘겨줄 props가 있을 때(우리 버킷리스트 앱은 App.js에서 list를 props로 넘겨주죠! 그럴 땐 이렇게 쓰면 됩니다!)

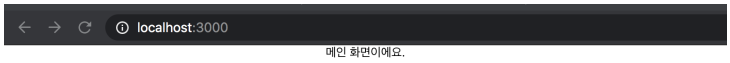
```
<Route path="주소[/home 처럼 /와 주소를 적어요]" render={(props) => (<BucketList list={this.state.list} />)} />
```

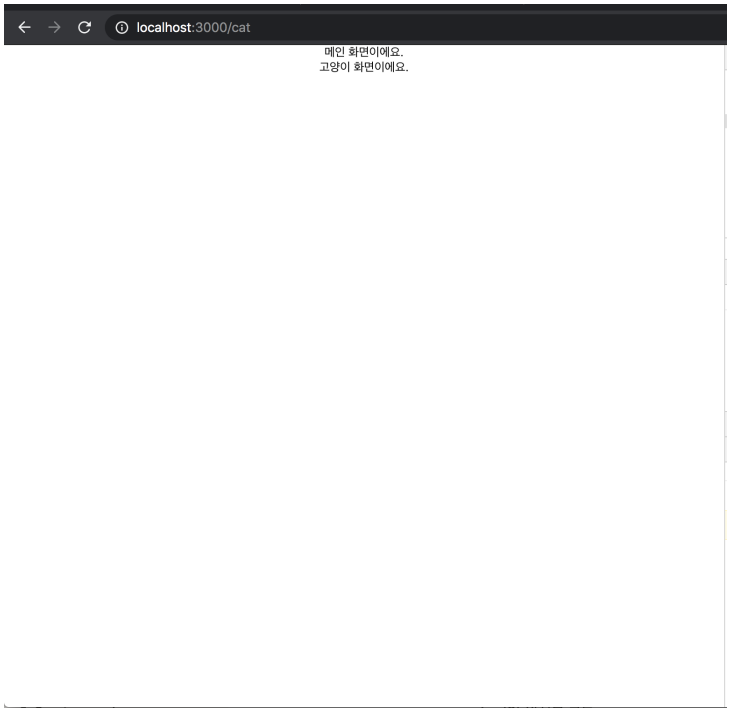
- App.js에 적용해보자

```
import React from 'react';
import logo from './logo.svg';
import './App.css';
// Route를 먼저 불러와줍니다.
import { Route } from "react-router-dom";

// 세부 페이지가 되어줄 컴포넌트들도 불러와주고요!
import Home from "./Home";
import Cat from "./Cat";
import Dog from "./Dog";

class App extends React.Component {

  constructor(props){
    super(props);
    this.state={};
  }

  render(){
    return (
      <div className="App">
        {/* 실제로 연결해볼까요! */}
        <Route path="/" component={Home} />
        <Route path="/cat" component={Cat} />
        <Route path="/dog" component={Dog} />
      </div>
    );
  }
}
```

```
export default App;
```

- 주소창에 "/", "/cat", "/dog"을 입력해보자!





😖 주소에 /를 입력했을 때는 Home컴포넌트만 뜨는데 왜 /cat에서는 Home과 Cat이 다 뜨는걸까요? 정답은 다음 항목에 있어요!

▼ (4) exact 적용하기

- 화면을 확인해봤더니, /cat과 /dog에서는 자꾸 Home 컴포넌트가 같이 나오죠?

> 👉 왜냐하면 "/" 이 기호가 "/cat"과 "/dog"에도 포함되어 있어서 그렇습니다!
> 아래처럼 exact를 추가하고 다시 주소를 이동해 봐요.

```
import React from 'react';
import logo from './logo.svg';
import './App.css';
// Route를 먼저 불러와줍니다.
import { Route } from "react-router-dom";

// 세부 페이지가 되어줄 컴포넌트들도 불러와주고요!
import Home from "./Home";
import Cat from "./Cat";
import Dog from "./Dog";

class App extends React.Component {

  constructor(props){
    super(props);
    this.state={};
  }

  render(){
    return (
      <div className="App">

        {/* 실제로 연결해볼까요! */}
        <Route path="/" exact component={Home} />
        <Route path="/cat" component={Cat} />
        <Route path="/dog" component={Dog} />
      </div>
    );
  }
}

export default App;
```

▼ (5) URL 파라미터사용하기

- 웹사이트 주소에는 파라미터와 쿼리라는 게 있어요. 우리는 그 중 파라미터 사용법을 알아볼 거예요!
- 이렇게 생겼어요!
  → 파라미터: /cat/nabi
  → 쿼리: /cat?name=nabi
- 파라미터 주는 방법

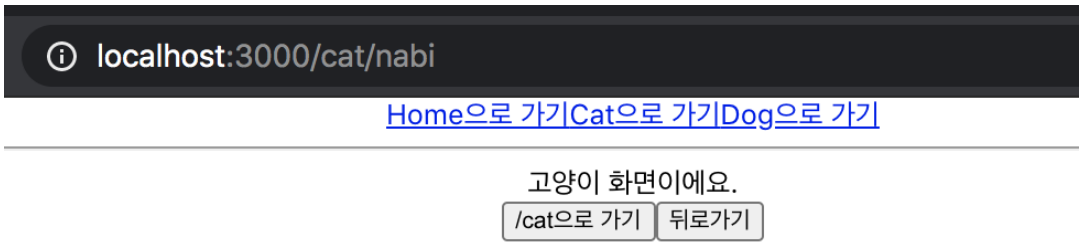
```
//App.js
...
// 파라미터 주기
<Route path="/cat/:cat_name" component={Cat}/>

...
```

- 파라미터 사용 방법

```
//Cat.js
import React from "react";

const Cat = (props) => {

    console.log(props.match);
```

```
    return (
        <div>고양이 화면이에요.</div>
    )
}

export default Cat;
```

- /cat/nabi로 주소를 이동해서 콘솔에 파라미터가 어떻게 찍히나 확인해봅시다!




```
▼{path: "/cat/:cat_name", url: "/cat/nabi", isExact: true, params: {…}}   Cat.js:5
    isExact: true
  ▶ params: {cat_name: "nabi"}
    path: "/cat/:cat_name"
    url: "/cat/nabi"
  ▶ __proto__: Object
```

▼ (6) 링크 이동 시키기

👉 매번 주소창을 찍고 페이지를 돌아다닐 순 없겠죠! react-router-dom으로 페이지를 이동하는 방법을 알아봅시다!

▼ -1) <Link/> 사용하기

- <Link/> 사용 방법

  링크 컴포넌트는 html 중 a 태그와 비슷한 역할을 해요. 리액트 내에서 페이지 전환을 도와줍니다.

```
<Link to="주소">[텍스트]</Link>
```

- App.js에 메뉴를 넣어보자!

  → 우리가 만든 메뉴처럼 <route> 바깥에 있는 돔요소는 페이지가 전환되어도 그대로 유지됩니다. (편리하죠!)

```
import React from 'react';
import logo from './logo.svg';
import './App.css';
// Route를 먼저 불러와줍니다.
// Link 컴포넌트도 불러왔어요.
import { Route, Link } from "react-router-dom";

// 세부 페이지가 되어줄 컴포넌트들도 불러와주고요!
import Home from "./Home";
import Cat from "./Cat";
import Dog from "./Dog";

class App extends React.Component {
```

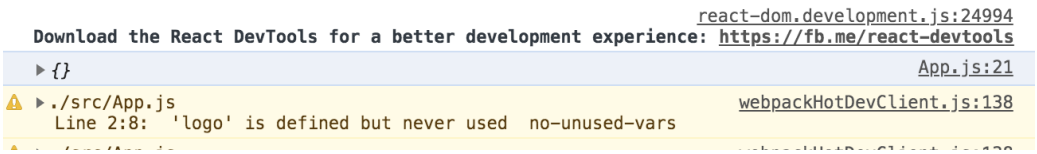
```
  constructor(props){
    super(props);
    this.state={};
  }

  render(){
    return (
      <div className="App">
        <div>
          <Link to="/">Home으로 가기</Link>
          <Link to="/cat">Cat으로 가기</Link>
          <Link to="/dog">Dog으로 가기</Link>
        </div>

        <hr />
        {/* 실제로 연결해볼까요! */}
        <Route path="/" exact component={Home} />
        <Route path="/cat" component={Cat} />
        <Route path="/dog" component={Dog} />
      </div>
    );
  }
}

export default App;
```

▼ -2) history 사용하기

> 👉 Link 컴포넌트를 클릭하지 않고, 조금 더 함수를 사용한 페이지 전환 방법을 알아봅시다!

- 먼저, App.js의 componentDidMount()에서 props를 콘솔로 찍어볼까요?

```
...
  componentDidMount(){
    console.log(this.props);
  }
...
```



→ 앗, props에 아무것도 없어요!

- withRouter를 설정하고 다시 props를 봅시다 😉

  → history 객체를 props로 받아오려면, withRouter를 설정해줘야해요.

```
import React from 'react';
import logo from './logo.svg';
import './App.css';
import { withRouter } from "react-router";

// Route를 먼저 불러와줍니다.
// Link 컴포넌트도 불러왔어요.
import { Route, Link } from "react-router-dom";

// 세부 페이지가 되어줄 컴포넌트들도 불러와주고요!
import Home from "./Home";
import Cat from "./Cat";
import Dog from "./Dog";

class App extends React.Component {
```

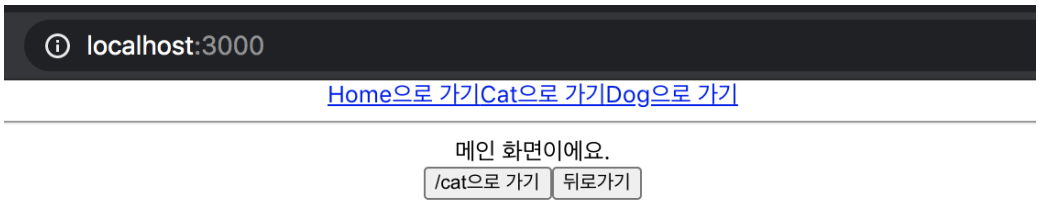
```
...
}
// 내보내는 부분에서 withRouter로 감싸줍니다
export default withRouter(App);
```

```
[HMR] Waiting for update signal from WDS...                    log.js:24
                                                  react-dom.development.js:24994
Download the React DevTools for a better development experience: https://fb.me/react-devtools
▶ {history: {…}, location: {…}, match: {…}, staticContext: undefined}   App.js:22
⚠ ▶ ./src/App.js                                          webpackHotDevClient.js:138
  Line 2:8:  'logo' is defined but never used  no-unused-vars
›
```

→ 위 화면처럼 잘 나오면 성공!

- 그럼 이제 버튼을 만들고, 페이지를 이동해봅시다!

  → /cat으로 가기, 뒤로가기 버튼 2개를 만들어볼거예요.




```
import React from 'react';
import logo from './logo.svg';
import './App.css';
import { withRouter } from "react-router";

// Route를 먼저 불러와줍니다.
// Link 컴포넌트도 불러왔어요.
import { Route, Link } from "react-router-dom";

// 세부 페이지가 되어줄 컴포넌트들도 불러와주고요!
import Home from "./Home";
import Cat from "./Cat";
import Dog from "./Dog";

class App extends React.Component {

  constructor(props){
    super(props);
    this.state={};
  }

  componentDidMount(){
    console.log(this.props);
  }

  render(){
    return (
      <div className="App">
        <div>
          <Link to="/">Home으로 가기</Link>
          <Link to="/cat">Cat으로 가기</Link>
          <Link to="/dog">Dog으로 가기</Link>
        </div>

        <hr />
        {/* 실제로 연결해볼까요! */}
        <Route path="/" exact component={Home} />
        <Route path="/cat" component={Cat} />
        <Route path="/dog" component={Dog} />
```

```
        <button onClick={() => {
          // props에 있는 history를 사용합니다.
          // push([이동할 주소])는 페이지를 이동시켜 줍니다.
          this.props.history.push('/cat');
        }}>
          /cat으로 가기
        </button>
        <button onClick={()=>{
          // goBack()은 뒤로가기 예요.
          this.props.history.goBack();
        }}>뒤로가기
        </button>
      </div>
    );
  }
}
// 내보내는 부분에서 withRouter로 감싸줍니다
export default withRouter(App);
```

04. Quiz_버킷리스트 상세 페이지 만들고 이동시키기

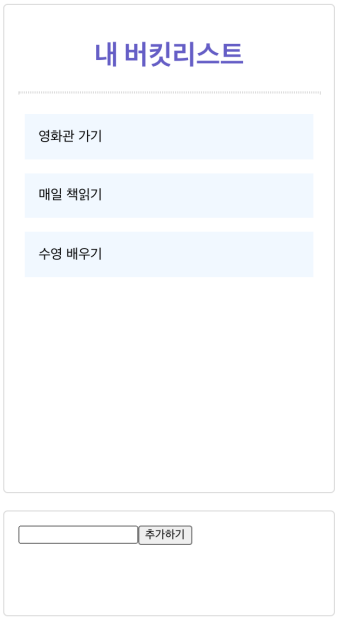
▼ 7) ✍️버킷리스트 상세페이지를 만들고 리스트 항목을 누르면 이동시키자!

> 🔥 버킷리스트 프로젝트 안에서도 react-router-dom을 설치해야합니다!

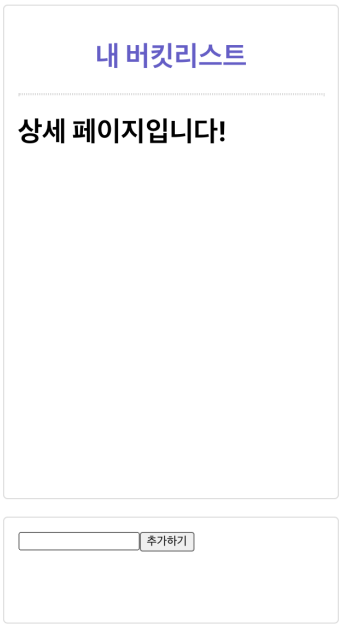
▼ Q. 퀴즈설명 : 버킷리스트 상세 페이지(/detail)을 만들고 이동시키기

> 💡 버킷리스트 항목이 나오는 부분만 route로 이동시켜주세요 🙂

▼ 모습 보기(메인페이지)



▼ 모습 보기(상세페이지)



▼ **[코드스니펫] - App.js**

```
import React from "react";
import logo from "./logo.svg";
// BucketList 컴포넌트를 import 해옵니다.
// import [컴포넌트 명] from [컴포넌트가 있는 파일경로];
import BucketList from "./BucketList";
import styled from "styled-components";

// 클래스형 컴포넌트는 이렇게 생겼습니다!
class App extends React.Component {
  constructor(props) {
    super(props);
    // App 컴포넌트의 state를 정의해줍니다.
    this.state = {
      list: ["영화관 가기", "매일 책읽기", "수영 배우기"],
    };
    // ref는 이렇게 선언합니다!
    this.text = React.createRef();
  }

  componentDidMount(){
    console.log(this.text);
  }

  addBucketList = () => {
    let list = this.state.list;
    const new_item = this.text.current.value;

    // 리액트에서는 concat으로 배열항목을 합쳐주는 게 좋아요. 아래처럼요!
    // list = list.concat(new_item);
    // this.setState({list: list});

    //  이건 가장 편한 배열 합치기 방법입니다.
    //  ...은 배열 안에 있는 항목을 전부 꺼내서 늘어놓는다는 뜻입니다. (스프레드 문법이라고 불러요.)
    this.setState({list: [...list, new_item]});
  }

  // 랜더 함수 안에 리액트 엘리먼트를 넣어줍니다!
  render() {

    return (
      <div className="App">
        <Container>
          <Title>내 버킷리스트</Title>
          <Line />
          {/* 컴포넌트를 넣어줍니다. */}
```

```
          {/* <컴포넌트 명 [props 명]={넘겨줄 것(리스트, 문자열, 숫자, ...)}/> */}
          <BucketList list={this.state.list} />
        </Container>
        {/* 인풋박스와 추가하기 버튼을 넣어줬어요. */}
        <Input>
          <input type="text" ref={this.text} />
          <button onClick={this.addBucketList}>추가하기</button>
        </Input>
      </div>
    );
  }
}

const Input = styled.div`
  max-width: 350px;
  min-height: 10vh;
  background-color: #fff;
  padding: 16px;
  margin: 20px auto;
  border-radius: 5px;
  border: 1px solid #ddd;
`;

const Container = styled.div`
  max-width: 350px;
  min-height: 60vh;
  background-color: #fff;
  padding: 16px;
  margin: 20px auto;
  border-radius: 5px;
  border: 1px solid #ddd;
`;

const Title = styled.h1`
  color: slateblue;
  text-align: center;
`;

const Line = styled.hr`
  margin: 16px 0px;
  border: 1px dotted #ddd;
`;

export default App;
```

▼ [코드스니펫] - BucketList.js

```
// 리액트 패키지를 불러옵니다.
import React from "react";
import styled from "styled-components";

const BucketList = (props) => {
  console.log(props);
  const my_lists = props.list;

  return (
    <ListStyle>
      {my_lists.map((list, index) => {
        return (
          <ItemStyle className="list_item" key={index} onClick={() => {}}>
            {list}
          </ItemStyle>
        );
      })}
    </ListStyle>
  );
};

const ListStyle = styled.div`
  display: flex;
  flex-direction: column;
  height: 100%;
  overflow-x: hidden;
  overflow-y: auto;
```

```
`;

const ItemStyle = styled.div`
  padding: 16px;
  margin: 8px;
  background-color: aliceblue;
`;

export default BucketList;
```

힌트:
1. Detail.js라는 파일 하나를 만들고 <Detail/> 컴포넌트를 만드세요!
2. 어떤 버킷 리스트 항목을 눌러도 그 페이지로 가게 해볼거예요!
3. history를 써봐요! (history도 props로 넘겨줄 수 있어요)
4. 꼭 코드스니펫을 복사해서 써주세요.
5. 버킷리스트 컴포넌트에서는 useRef를 쓰는 대신 element에 직접 onClick을 줘서 해봅시다!

▼ A. 함께하기(완성본)

어때요, 할만했나요? 다만 조금씩 다른 방법으로 해결하셨더라도,
다음 강의 진행을 위해 아래 코드를 복사→붙여넣기 해주세요!

▼ [코드스니펫] - App.js

```
import React from "react";

import { withRouter } from "react-router";
import { Route } from "react-router-dom";

// import [컴포넌트 명] from [컴포넌트가 있는 파일경로];
import BucketList from "./BucketList";
import styled from "styled-components";
import Detail from "./Detail";

// 클래스형 컴포넌트는 이렇게 생겼습니다!
class App extends React.Component {
  constructor(props) {
    super(props);
    // App 컴포넌트의 state를 정의해줍니다.
    this.state = {
      list: ["영화관 가기", "매일 책읽기", "수영 배우기"],
    };
    // ref는 이렇게 선언합니다!
    this.text = React.createRef();
  }

  componentDidMount() {
    console.log(this.text);
  }

  addBucketList = () => {
    let list = this.state.list;
    const new_item = this.text.current.value;

    // 리액트에서는 concat으로 배열항목을 합쳐주는 게 좋아요. 아래처럼요!
    // list = list.concat(new_item);
    // this.setState({list: list});

    //  이건 가장 편한 배열 합치기 방법입니다.
    //  ...은 배열 안에 있는 항목을 전부 꺼내서 늘어놓는다는 뜻입니다. (스프레드 문법이라고 불러요.)
    this.setState({ list: [...list, new_item] });
  };
```

```
  // 랜더 함수 안에 리액트 엘리먼트를 넣어줍니다!
  render() {
    return (
      <div className="App">
        <Container>
          <Title>내 버킷리스트</Title>
          <Line />
          {/* 컴포넌트를 넣어줍니다. */}
          {/* <컴포넌트 명 [props 명]={넘겨줄 것(리스트, 문자열, 숫자, ...)}/> */}
          {/* Route 쓰는 법 2가지를 모두 써봅시다! */}
          <Route
            path="/"
            exact
            render={(props) => <BucketList list={this.state.list} history={this.props.history}/>}
          />
          <Route path="/detail" component={Detail}/>
        </Container>
        {/* 인풋박스와 추가하기 버튼을 넣어줬어요. */}
        <Input>
          <input type="text" ref={this.text} />
          <button onClick={this.addBucketList}>추가하기</button>
        </Input>
      </div>
    );
  }
}

const Input = styled.div`
  max-width: 350px;
  min-height: 10vh;
  background-color: #fff;
  padding: 16px;
  margin: 20px auto;
  border-radius: 5px;
  border: 1px solid #ddd;
`;

const Container = styled.div`
  max-width: 350px;
  min-height: 60vh;
  background-color: #fff;
  padding: 16px;
  margin: 20px auto;
  border-radius: 5px;
  border: 1px solid #ddd;
`;

const Title = styled.h1`
  color: slateblue;
  text-align: center;
`;

const Line = styled.hr`
  margin: 16px 0px;
  border: 1px dotted #ddd;
`;
// withRouter 적용
export default withRouter(App);
```

▼ [코드스니펫] - Detail.js

```
// 리액트 패키지를 불러옵니다.
import React from "react";

const Detail = (props) => {

  return <h1>상세 페이지입니다!</h1>;
};

export default Detail;
```

▼ [코드스니펫] - BucketList.js(완성)

```
// 리액트 패키지를 불러옵니다.
import React from "react";
import styled from "styled-components";

const BucketList = (props) => {
  console.log(props);
  const my_lists = props.list;

  return (
    <ListStyle>
      {my_lists.map((list, index) => {
        return (
          <ItemStyle className="list_item" key={index} onClick={() => {
          props.history.push('/detail');
          }}>
            {list}
          </ItemStyle>
        );
      })}
    </ListStyle>
  );
};

const ListStyle = styled.div`
  display: flex;
  flex-direction: column;
  height: 100%;
  overflow-x: hidden;
  overflow-y: auto;
`;

const ItemStyle = styled.div`
  padding: 16px;
  margin: 8px;
  background-color: aliceblue;
`;

export default BucketList;
```

05. 라우팅, 조금 더 꼼꼼히 쓰려면?

▼ 8) 잘못된 주소 처리하기

> 👉 exact로 중복 주소를 처리하는 방법은 이미 배웠습니다!
> 이번엔 우리가 미리 정하지 않은 주소로 들어온 경우를 다뤄볼게요.
> (저는 버킷리스트 프로젝트에서 진행합니다!)

- 일단 NotFound.js 파일을 만들고 빈 컴포넌트를 만들어주세요.

```
import React from "react";

const NotFound = (props) => {
  return <h1>주소가 올바르지 않아요!</h1>;
};

export default NotFound;
```

- App.js에서 불러옵니다.

```
import NotFound from "./NotFound";
```

- Switch를 추가해주고,

```
...
import { Route, Switch } from "react-router-dom";
...
class App extends React.Component {
  ...
  render() {
    return (
      <div className="App">
        ...
          <Switch>
            <Route
              path="/"
              exact
              render={(props) => (
                <BucketList
                  list={this.state.list}
                  history={this.props.history}
                />
              )}
            />
            <Route path="/detail" component={Detail} />
          </Switch>
        ...
      </div>
    );
  }
}
...
```

- NotFound컴포넌트를 Route에 주소 없이 연결하면 끝!

```
...
          <Switch>
            <Route
              path="/"
              exact
              render={(props) => (
                <BucketList
                  list={this.state.list}
                  history={this.props.history}
                />
              )}
            />
            <Route path="/detail" component={Detail} />
            <Route component={NotFound} />
          </Switch>
...
```

▼ 9) ✍️<NotFound/>에 뒤로가기 버튼을 달아보자!

> 👉 뒤로가기 버튼, 한 번 달아봤던겁니다! 잠깐 일시정지하시고 뒤로가기 버튼을 달아봐요!

- App.js에서 history를 props로 넘겨주는 게 먼저!

```
<Route render={(props) => (
                <NotFound
                  history={this.props.history}
                />
              )}/>
```

- NotFound.js에서는,

```
import React from "react";

const NotFound = (props) => {
  return (
    <div>
      <h1>주소가 올바르지 않아요!</h1>
      <button>뒤로가기</button>
    </div>
  );
};

export default NotFound;
```

- props를 받아서,

```
const NotFound = (props) => {
  console.log(props);
...
```

- onClick에 넣어주기!

```
<button onClick={() => {props.history.goBack();}}>뒤로가기</button>
```

06. 리덕스란?

▼ 10) 리덕스 패키지 설치 & 공식문서 보기

> 🔥 리덕스는 아주 흔히 사용하는 상태관리 라이브러리입니다.
> 전역 상태관리를 편히 할 수 있게 해주는 고마운 친구죠!
> (버킷리스트 프로젝트에서 이어할거예요!)

- 리덕스 패키지 설치하기

```
yarn add redux react-redux
```

- 공식문서 보러가기→

▼ 11) 리덕스 개념과 용어

> 👉 **리덕스는 데이터를 한 군데 몰아넣고, 여기저기에서 꺼내볼 수 있게 해주는 친구**입니다.
> 아래 용어들은 리덕스의 기본 용어인데, 여러분이 키워드 삼기 좋은 용어들이에요. 앞으로 자주 볼 단어들이니 미리 친해집시다!

▼ (1) State

👉 리덕스에서는 저장하고 있는 상태값("데이터"라고 생각하셔도 돼요!)를 state라고 불러요.
딕셔너리 형태({[key]: value})형태로 보관합니다.

▼ (2) Action

👉 상태에 변화가 필요할 때(=가지고 있는 데이터를 변경할 때) 발생하는 것입니다.

```
// 액션은 객체예요. 이런 식으로 쓰여요. type은 이름같은 거예요! 저희가 정하는 임의의 문자열을 넣습니다.
{type: 'CHANGE_STATE', data: {...}}
```

▼ (3) ActionCreator

👉 액션 생성 함수라고도 부릅니다. 액션을 만들기 위해 사용합니다.

```
//이름 그대로 함수예요!
const changeState = (new_data) => {
// 액션을 리턴합니다! (액션 생성 함수니까요. 제가 너무 당연한 이야기를 했나요? :))
  return {
    type: 'CHANGE_STATE',
    data: new_data
  }
}
```

▼ (4) Reducer

👉 리덕스에 저장된 상태(=데이터)를 변경하는 함수입니다.
우리가 액션 생성 함수를 부르고 → 액션을 만들면 → 리듀서가 현재 상태(=데이터)와 액션 객체를 받아서 → 새로운 데이터를 만들고 → 리턴해줍니다.

```
// 기본 상태값을 임의로 정해줬어요.
const initialState = {
  name: 'mean0'
}

function reducer(state = initialState, action) {
  switch(action.type){

    // action의 타입마다 케이스문을 걸어주면,
    // 액션에 따라서 새로운 값을 돌려줍니다!
    case CHANGE_STATE:
      return {name: 'mean1'};

    default:
      return false;
  }
}
```

▼ (5) Store

👉 우리 프로젝트에 리덕스를 적용하기 위해 만드는 거예요!
스토어에는 리듀서, 현재 애플리케이션 상태, 리덕스에서 값을 가져오고 액션을 호출하기 위한 몇 가지 내장 함수가 포함되어 있습니다.
생김새는 딕셔너리 혹은 json처럼 생겼어요.
내장함수를 어디서 보냐구요? → 공식문서에서요! 😉

▼ (6) dispatch

👉 디스패치는 우리가 앞으로 정말 많이 쓸 스토어의 내장 함수예요!
액션을 발생 시키는 역할을 합니다.

```
// 실제로는 이것보다 코드가 길지만,
// 간단히 표현하자면 이런 식으로 우리가 발생시키고자 하는 액션을 파라미터로 넘겨서 사용합니다.
dispatch(action);
```

💡 몰라도 되는, 하지만 알면 재미있는 이야기
리덕스는 사실, 리액트와 별도로 사용할 수 있는 친구입니다. 상태관리를 위해 다른 프론트엔드 프레임워크/라이브러리와 함께 쓸 수 있어요.

▼ 12) 리덕스의 3가지 특징

👉 눈으로 쭉 읽어보세요! 당장 이해하지 못해도 괜찮아요. 프로젝트를 끝낼 쯤엔 이게 그런 소리였구나 하실 겁니다. 😎

▼ (1) store는 1개만 쓴다!

👉 리덕스는 단일 스토어 규칙을 따릅니다. 한 프로젝트에 스토어는 하나만 씁니다.

▼ (2) store의 state(데이터)는 오직 action으로만 변경할 수 있다!

👉 리액트에서도 state는 setState()나, useState() 훅을 써서만 변경 가능했죠!
데이터가 마구잡이로 변하지 않도록 **불변성**을 유지해주기 위함입니다.
불변성 뭐냐구요? 간단해요! 허락없이 데이터가 바뀌면 안된단 소리입니다!

조금 더 그럴 듯하게 말하면, 리덕스에 저장된 데이터 = 상태 = state는 **읽기 전용**입니다.

그런데... 액션으로 변경을 일으킨다면서요? 리듀서에서 변한다고 했잖아요?
→ 네, 그것도 맞아요. 조금 더 정확히 해볼까요!
가지고 있던 값을 수정하지 않고, 새로운 값을 만들어서 상태를 갈아끼웁니다!
즉, A에 +1을 할 때,
A = A+1이 되는 게 아니고, A' = A+1이라고 새로운 값을 만들고 A를 A'로 바꾸죠.

▼ (3) 어떤 요청이 와도 리듀서는 같은 동작을 해야한다!

👉 **리듀서는 순수한 함수여야 한다**는 말입니다.
순수한 함수라는 건,
- 파라미터 외의 값에 의존하지 않아야하고,
- 이전 상태는 수정하지(=건드리지) 않는다. (변화를 준 새로운 객체를 return 해야합니다.)
- 파라미터가 같으면, 항상 같은 값을 반환
- 리듀서는 이전 상태와 액션을 파라미터로 받는다.

07. 리덕스를 통한 리액트 상태관리

▼ 13) 상태관리! 왜 필요할까?

👉 [복습하자!]
저희가 만들었던 버킷리스트를 되짚어봅시다!
지금은 App.js에서 리스트 항목 배열을 넣어두고, props로 넘겨주고 있습니다.
그리고 추가하기 버튼도 App.js에 있고요.

만약에, 우리가 이 추가하기 버튼과 텍스트 영역을 AddListItem 컴포넌트를 만들어 분리하고 싶다면 어떻게 해야할 것 같나요?

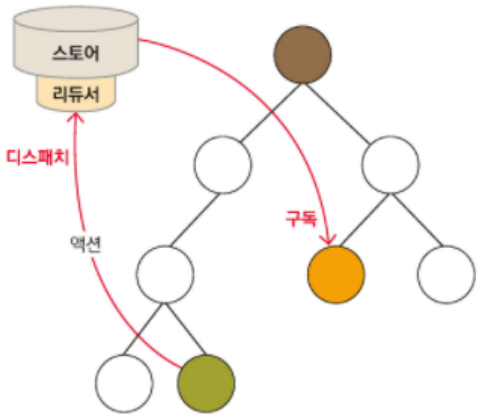
파일을 만들고 코드를 만들면 될까요?
그렇게 하면 추가하기 버튼을 눌렀을 때 정말 App 컴포넌트의 state를 수정할 수 있을까요? 😢
→ 네, 자식 컴포넌트는 부모 컴포넌트의 state를 맘대로 조작할 수 없어요.
→ 왜냐면 데이터는 부모에서 자식으로 흐르게 하기로 했으니까요.(데이터는 단방향으로!)

그런데 만약에, App 컴포넌트와 AddListItem 컴포넌트가 같은 데이터 저장소를 본다면 어떨까요?
→ AddListItem에서 추가를 하면 App이 보고 있는 데이터도 같이 추가가 되겠죠!

🔥 리덕스는 여러 컴포넌트가 동일한 상태를 보고 있을 때 굉장히 유용합니다!
또, 데이터를 관리하는 로직을 컴포넌트에서 빼면, 컴포넌트는 정말 뷰만 관리할 수 있잖아요!
코드가 깔끔해질테니, 유지보수에도 아주 좋겠죠. 🙂

▼ 14) 상태관리 흐름을 알아보자!

👉 [상태관리 흐름도]
딱 4가지만 알면 됩니다! Store, Action, Reducer, 그리고 Component!
아주 큰 흐름만 잘 파악해도 굳굳!



- (1) 리덕스 Store를 Component에 연결한다.
- (2) Component에서 상태 변화가 필요할 때 Action을 부른다.
- (3) Reducer를 통해서 새로운 상태 값을 만들고,
- (4) 새 상태값을 Store에 저장한다.
- (5) Component는 새로운 상태값을 받아온다. (props를 통해 받아오니까, 다시 랜더링 되겠죠?)

08. 리덕스 써보기

▼ 15) 덕스(ducks) 구조

- 보통 리덕스를 사용할 때는, 모양새대로 action, actionCreator, reducer를 분리해서 작성합니다.
  (액션은 액션끼리, 액션생성함수는 액션생성함수끼리, 리듀서는 리듀서끼리 작성합니다.)
- 덕스 구조는 모양새로 묶는 대신 기능으로 묶어 작성합니다.
  (버킷리스트를 예로 들자면, 버킷리스트의 action, actionCreator, reducer를 한 파일에 넣는 거예요.)
- 우리는 덕스 구조로 리덕스 모듈을 만들어볼거예요!

> 👉 [외울 필요 없어요!]
> 덕스 구조를 잘 설명해 주는 사이트가 있습니다. 😉 헷갈리실 때 들어가서 읽어보시면 되고, 모듈을 새로 만들 때 복사해서 쓰셔도 좋습니다.
> 사이트 바로가기→

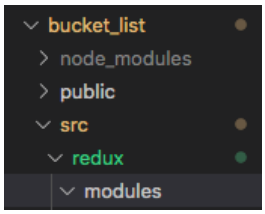
▼ [코드스니펫] - 리덕스 모듈 예제

```
// widgets.js

// Actions
const LOAD   = 'my-app/widgets/LOAD';
const CREATE = 'my-app/widgets/CREATE';
const UPDATE = 'my-app/widgets/UPDATE';
const REMOVE = 'my-app/widgets/REMOVE';

// Reducer
export default function reducer(state = {}, action = {}) {
  switch (action.type) {
    // do reducer stuff
    default: return state;
  }
}
```

```
// Action Creators
export function loadWidgets() {
  return { type: LOAD };
}

export function createWidget(widget) {
  return { type: CREATE, widget };
}

export function updateWidget(widget) {
  return { type: UPDATE, widget };
}

export function removeWidget(widget) {
  return { type: REMOVE, widget };
}

// side effects, only as applicable
// e.g. thunks, epics, etc
export function getWidget () {
  return dispatch => get('/widget').then(widget => dispatch(updateWidget(widget)))
}
```

▼ 16) 첫번째 모듈 만들기

👉 일단 폴더부터 만들어요!
src 폴더 아래에 redux라는 폴더를 만들고, 그 안에 modules라는 폴더를 만들어주세요.
modules 아래에 bucket.js라는 파일을 만들고 리덕스 모듈 예제를 붙여넣어주세요.
아래 과정을 하나하나 밟으며 **버킷리스트 항목을 리덕스에서 관리하도록** 고쳐봅시다!



▼ (1) Action

👉 우리는 지금 버킷리스트를 가져오는 것, 생성하는 것 2가지 변화가 있죠?
두 가지 액션을 만듭시다.

```
const LOAD = 'bucket/LOAD';
const CREATE = 'bucket/CREATE';
```

▼ (2) initialState

👉 초기 상태값을 만들어줄거예요! 그러니까, 기본 값이죠.

```
const initialState = {
  list: ["영화관 가기", "매일 책읽기", "수영 배우기"],
};
```

▼ (3) Action Creactor

👉 액션 생성 함수를 작성합니다.

```
export const loadBucket = (bucket) => {
    return { type: LOAD, bucket };
}

export const createBucket = (bucket) => {
    return {type: CREATE, bucket};
}
```

▼ (4) Reducer

👉 리듀서를 작성합니다.
load할 땐, 가지고 있던 기본값을 그대로 뿌려주면 되겠죠?
create할 땐, 새로 받아온 값을 가지고 있던 값에 더해서 리턴해주면 될거예요!
(우리는 action으로 넘어오는 bucket이 text값인 걸 알고 있죠! 이미 추가해봤잖아요.)

```
export default function reducer(state = initialState, action = {}) {
  switch (action.type) {
    // do reducer stuff
    case "bucket/LOAD":
      return state;

    case "bucket/CREATE":
      const new_bucket_list = [...state.list, action.bucket];
      return { list: new_bucket_list };

    default:
      return state;
  }
}
```

▼ (5) Store

👉 redux 폴더 하위에 configStore.js 파일을 만들고 스토어를 만들어볼게요!

```
//configStore.js
import { createStore, combineReducers } from "redux";
import bucket from './modules/bucket';
import { createBrowserHistory } from "history";

// 브라우저 히스토리를 만들어줍니다.
export const history = createBrowserHistory();
// root 리듀서를 만들어줍니다.
// 나중에 리듀서를 여러개 만들게 되면 여기에 하나씩 추가해주는 거예요!
const rootReducer = combineReducers({ bucket });

// 스토어를 만듭니다.
const store = createStore(rootReducer);

export default store;
```

09. 리덕스와 컴포넌트를 연결하자!

▼ 17) Store 연결하기

👉 store를 다 만들었으니 이젠 연결할 차례! (끝이 다와가요!)
index.js에서 필요한 작업을 해줄거예요.
스토어를 불러오고 → 우리 버킷리스트에 주입하면 끝!

▼ (1) index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';
import {BrowserRouter} from 'react-router-dom';

// 우리의 버킷리스트에 리덕스를 주입해줄 프로바이더를 불러옵니다!
import { Provider } from "react-redux";
// 연결할 스토어도 가지고 와요.
import store from "./redux/configStore";
ReactDOM.render(
  <Provider store={store}>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </Provider>,
  document.getElementById("root")
);

// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

10. 컴포넌트에서 리덕스 데이터 사용하기

▼ 18) 컴포넌트에서 리덕스 액션 사용하는 법

👉 이제 우리 리덕스가 버킷리스트에 붙었는데, 아직 실감이 안나죠?
실감이 나도록, App 컴포넌트에 붙은 state를 리덕스로 교체해볼까요?

▼ (1) 클래스형 컴포넌트에서 리덕스 데이터 사용하기

- -1) 리덕스 모듈과 connect 함수를 불러옵니다.
- -2) 상태값을 가져오는 함수와 액션 생성 함수를 부르는 함수를 만들어줍니다.
- -3) connect로 컴포넌트와 스토어를 엮어줍니다.
- -4) 콘솔에 this.props를 찍어봅니다. (스토어에 있는 값이 잘 나왔는지 볼까요!)
- -5) this.state에 있는 list를 지우고 스토어에 있는 값으로 바꿔봅시다!
- -6) setState를 this.props.create로 바꿔봅시다!

▼ **[코드스니펫] - App.js**

```
import React from "react";
```

```
import { withRouter } from "react-router";
import { Route, Switch } from "react-router-dom";

// import [컴포넌트 명] from [컴포넌트가 있는 파일경로];
import BucketList from "./BucketList";
import styled from "styled-components";
import Detail from "./Detail";
import NotFound from "./NotFound";

// 리덕스 스토어와 연결하기 위해 connect라는 친구를 호출할게요!
import {connect} from 'react-redux';
// 리덕스 모듈에서 (bucket 모듈에서) 액션 생성 함수 두개를 가져올게요!
import {loadBucket, createBucket} from './redux/modules/bucket';

// 이 함수는 스토어가 가진 상태값을 props로 받아오기 위한 함수예요.
const mapStateTopProps = (state) => ({
  bucket_list: state.bucket.list,
});

// 이 함수는 값을 변화시키기 위한 액션 생성 함수를 props로 받아오기 위한 함수예요.
const mapDispatchToProps = (dispatch) => ({
  load: () => {
    dispatch(loadBucket());
  },
  create: (new_item) => {
    dispatch(createBucket(new_item));
  }
});

// 클래스형 컴포넌트는 이렇게 생겼습니다!
class App extends React.Component {
  constructor(props) {
    super(props);
    // App 컴포넌트의 state를 정의해줍니다.
    this.state = {

    };
    // ref는 이렇게 선언합니다!
    this.text = React.createRef();
  }

  componentDidMount() {
    console.log(this.props);
  }

  addBucketList = () => {
    const new_item = this.text.current.value;
    this.props.create(new_item);
  };

  // 랜더 함수 안에 리액트 엘리먼트를 넣어줍니다!
  render() {
    return (
      <div className="App">
        <Container>
          <Title>내 버킷리스트</Title>
          <Line />
          {/* 컴포넌트를 넣어줍니다. */}
          {/* <컴포넌트 명 [props 명]={넘겨줄 것(리스트, 문자열, 숫자, ...)}/> */}
          {/* Route 쓰는 법 2가지를 모두 써봅시다! */}
          <Switch>
            <Route
              path="/"
              exact
              render={(props) => (
                <BucketList
                  list={this.props.bucket_list}
                  history={this.props.history}
                />
              )}
            />
            <Route path="/detail" component={Detail} />
            <Route
              render={(props) => <NotFound history={this.props.history} />}
            />
          </Switch>
        </Container>
```

```
        {/* 인풋박스와 추가하기 버튼을 넣어줬어요. */}
        <Input>
          <input type="text" ref={this.text} />
          <button onClick={this.addBucketList}>추가하기</button>
        </Input>
      </div>
    );
  }
}

const Input = styled.div`
  max-width: 350px;
  min-height: 10vh;
  background-color: #fff;
  padding: 16px;
  margin: 20px auto;
  border-radius: 5px;
  border: 1px solid #ddd;
`;

const Container = styled.div`
  max-width: 350px;
  min-height: 60vh;
  background-color: #fff;
  padding: 16px;
  margin: 20px auto;
  border-radius: 5px;
  border: 1px solid #ddd;
`;

const Title = styled.h1`
  color: slateblue;
  text-align: center;
`;

const Line = styled.hr`
  margin: 16px 0px;
  border: 1px dotted #ddd;
`;
// withRouter 적용
// connect로 묶어줬습니다!
export default connect(mapStateTopProps, mapDispatchToProps)(withRouter(App));
```

▼ (2) 함수형 컴포넌트에서 리덕스 데이터 사용하기

👉 리덕스도 리액트처럼 훅이 있어요. 훅을 사용해서 액션 생성 함수를 불러보고, 스토어에 저장된 값도 가져와 볼게요.

▼ -1) BucketList.js에 useSelector() 적용하기

```
// 리액트 패키지를 불러옵니다.
import React from "react";
import styled from "styled-components";

// redux hook을 불러옵니다.
import {useDispatch, useSelector} from 'react-redux';

const BucketList = (props) => {
  // 버킷리스트를 리덕스 훅으로 가져오기
  const bucket_list = useSelector(state => state.bucket.list);

  console.log(bucket_list);

  return (
    <ListStyle>
      {bucket_list.map((list, index) => {
        return (
          <ItemStyle
            className="list_item"
            key={index}
```

```
            onClick={() => {
              props.history.push("/detail");
            }}
          >
            {list}
          </ItemStyle>
        );
      })}
    </ListStyle>
  );
};

const ListStyle = styled.div`
  display: flex;
  flex-direction: column;
  height: 100%;
  overflow-x: hidden;
  overflow-y: auto;
`;

const ItemStyle = styled.div`
  padding: 16px;
  margin: 8px;
  background-color: aliceblue;
`;

export default BucketList;
```

▼ -2) 몇 번째 상세에 와있는 지 알기 위해, URL 파라미터를 적용하자

```
//App.js
...
          <Switch>
            <Route
              path="/"
              exact
              render={(props) => <BucketList history={this.props.history} />}
            />
            <Route
              path="/detail/:index"
              render={(props) => <Detail history={this.props.history} />}
            />
            <Route
              render={(props) => <NotFound history={this.props.history} />}
            />
          </Switch>
...
```

```
//BucketList.js
...
      {bucket_list.map((list, index) => {
        return (
          <ItemStyle
            className="list_item"
            key={index}
            onClick={() => {
              // 배열의 몇번째 항목을 눌렀는 지, url 파라미터로 넘겨줍니다.
              props.history.push("/detail"+index);
            }}
          >
            {list}
          </ItemStyle>
        );
      })}
    </ListStyle>
  );
};
...
```

▼ -3) 상세페이지에서 버킷리스트 내용을 띄워보자

```
//Detail.js
// 리액트 패키지를 불러옵니다.
import React from "react";

// redux hook을 불러옵니다.
import { useDispatch, useSelector } from "react-redux";

const Detail = (props) => {
  // 스토어에서 상태값 가져오기
  const bucket_list = useSelector((state) => state.bucket.list);
  // url 파라미터에서 인덱스 가져오기
  let bucket_index = parseInt(props.match.params.index);

  return <h1>{bucket_list[bucket_index]}</h1>;
};

export default Detail;
```

11. Quiz_버킷 리스트 데이터를 삭제해보기

▼ 19) ✍️버킷 리스트 상세에서 삭제 버튼을 추가하고 리덕스에서 빼보자

▼ Q. 퀴즈설명: 상세페이지에서 삭제 버튼을 두고, 항목을 삭제해보자!

▼ [코드스니펫] - Detail.js

```
// 리액트 패키지를 불러옵니다.
import React from "react";

// redux hook을 불러옵니다.
import { useDispatch, useSelector } from "react-redux";
// 내가 만든 액션 생성 함수를 불러옵니다.
// import {deleteBucket} from "./redux/modules/bucket";

const Detail = (props) => {
    // const dispatch = useDispatch();


  // 스토어에서 상태값 가져오기
  const bucket_list = useSelector((state) => state.bucket.list);
  // url 파라미터에서 인덱스 가져오기
  let bucket_index = parseInt(props.match.params.index);

  return (
    <div>
      <h1>{bucket_list[bucket_index]}</h1>
      <button onClick={() => {
        //   dispatch(); <- 괄호안에는 액션 생성 함수가 들어가야겠죠?
        // 예를 들면 이렇게요.
        // dispatch(deleteBucket(bucket_index));
      }}>삭제하기</button>
    </div>
  );
};

export default Detail;
```

▼ [코드스니펫] - BucketList.js

```
// 리액트 패키지를 불러옵니다.
import React from "react";
import styled from "styled-components";
```

```
// redux hook을 불러옵니다.
import {useDispatch, useSelector} from 'react-redux';

const BucketList = (props) => {
  // 버킷리스트를 리덕스 훅으로 가져오기
  const bucket_list = useSelector(state => state.bucket.list);

  console.log(bucket_list);

  return (
    <ListStyle>
      {bucket_list.map((list, index) => {
        return (
          <ItemStyle
            className="list_item"
            key={index}
            onClick={() => {
              // 배열의 몇번째 항목을 눌렀는 지, url 파라미터로 넘겨줍니다.
              props.history.push("/detail/"+index);
            }}
          >
            {list}
          </ItemStyle>
        );
      })}
    </ListStyle>
  );
};

const ListStyle = styled.div`
  display: flex;
  flex-direction: column;
  height: 100%;
  overflow-x: hidden;
  overflow-y: auto;
`;

const ItemStyle = styled.div`
  padding: 16px;
  margin: 8px;
  background-color: aliceblue;
`;

export default BucketList;
```

▼ [코드스니펫] - App.js

```
import React from "react";

import { withRouter } from "react-router";
import { Route, Switch } from "react-router-dom";

// import [컴포넌트 명] from [컴포넌트가 있는 파일경로];
import BucketList from "./BucketList";
import styled from "styled-components";
import Detail from "./Detail";
import NotFound from "./NotFound";

// 리덕스 스토어와 연결하기 위해 connect라는 친구를 호출할게요!
import {connect} from 'react-redux';
// 리덕스 모듈에서 (bucket 모듈에서) 액션 생성 함수 두개를 가져올게요!
import {loadBucket, createBucket} from './redux/modules/bucket';

// 이 함수는 스토어가 가진 상태값을 props로 받아오기 위한 함수예요.
const mapStateTopProps = (state) => ({
  bucket_list: state.bucket.list,
});

// 이 함수는 값을 변화시키기 위한 액션 생성 함수를 props로 받아오기 위한 함수예요.
const mapDispatchToProps = (dispatch) => ({
  load: () => {
    dispatch(loadBucket());
  },
```

```
  create: (new_item) => {
    dispatch(createBucket(new_item));
  }
});

// 클래스형 컴포넌트는 이렇게 생겼습니다!
class App extends React.Component {
  constructor(props) {
    super(props);
    // App 컴포넌트의 state를 정의해줍니다.
    this.state = {

    };
    // ref는 이렇게 선언합니다!
    this.text = React.createRef();
  }

  componentDidMount() {
    console.log(this.props);
  }

  addBucketList = () => {
    const new_item = this.text.current.value;
    this.props.create(new_item);
  };

  // 랜더 함수 안에 리액트 엘리먼트를 넣어줍니다!
  render() {
    return (
      <div className="App">
        <Container>
          <Title>내 버킷리스트</Title>
          <Line />
          {/* 컴포넌트를 넣어줍니다. */}
          {/* <컴포넌트 명 [props 명]={넘겨줄 것(리스트, 문자열, 숫자, ...)}/> */}
          {/* Route 쓰는 법 2가지를 모두 써봅시다! */}
          <Switch>
            <Route path="/" exact component={BucketList} />
            <Route path="/detail/:index" component={Detail} />
            <Route component={NotFound} />
          </Switch>
        </Container>
        {/* 인풋박스와 추가하기 버튼을 넣어줬어요. */}
        <Input>
          <input type="text" ref={this.text} />
          <button onClick={this.addBucketList}>추가하기</button>
        </Input>
      </div>
    );
  }
}

const Input = styled.div`
  max-width: 350px;
  min-height: 10vh;
  background-color: #fff;
  padding: 16px;
  margin: 20px auto;
  border-radius: 5px;
  border: 1px solid #ddd;
`;

const Container = styled.div`
  max-width: 350px;
  min-height: 60vh;
  background-color: #fff;
  padding: 16px;
  margin: 20px auto;
  border-radius: 5px;
  border: 1px solid #ddd;
`;

const Title = styled.h1`
  color: slateblue;
  text-align: center;
`;
```

```
const Line = styled.hr`
  margin: 16px 0px;
  border: 1px dotted #ddd;
`;
// withRouter 적용
// connect로 묶어줬습니다!
export default connect(mapStateTopProps, mapDispatchToProps)(withRouter(App));
```

힌트:
1. 조건에 맞춰 배열 항목을 필터링 해주는 array의 내장함수 filter를 사용합니다.
(filter가 뭔지 모르신다면 검색해서 해보세요!)
→ url 파라미터가 배열의 index이니, 그 것만 빼고 나머지를 새로운 배열에 넣어주면 되겠네요!
2. 액션, 액션 생성 함수, 리듀서에 **삭제**에 관한 내용을 넣어줘야겠죠?
→ create를 참고해서 delete를 만들어보세요!
3. Detail.js에 제가 useDispatch() 사용법을 주석처리 해두었습니다 :) 주석을 참고해보세요!

▼ A. 함께하기(완성본)

▼ **[코드스니펫] - Detail.js**

```
// 리액트 패키지를 불러옵니다.
import React from "react";

// redux hook을 불러옵니다.
import { useDispatch, useSelector } from "react-redux";
// 내가 만든 액션 생성 함수를 불러옵니다.
import {deleteBucket} from "./redux/modules/bucket";

const Detail = (props) => {
    const dispatch = useDispatch();


  // 스토어에서 상태값 가져오기
  const bucket_list = useSelector((state) => state.bucket.list);
  // url 파라미터에서 인덱스 가져오기
  let bucket_index = parseInt(props.match.params.index);

  console.log(props);
  return (
    <div>
      <h1>{bucket_list[bucket_index]}</h1>
      <button onClick={() => {
        //   dispatch(); <- 괄호안에는 액션 생성 함수가 들어가야겠죠?
        // 예를 들면 이렇게요.
        dispatch(deleteBucket(bucket_index));
        props.history.goBack();
      }}>삭제하기</button>
    </div>
  );
};

export default Detail;
```

▼ **[코드스니펫] - BucketList.js**

```
// 리액트 패키지를 불러옵니다.
import React from "react";
import styled from "styled-components";

// redux hook을 불러옵니다.
import {useDispatch, useSelector} from 'react-redux';

const BucketList = (props) => {
  // 버킷리스트를 리덕스 훅으로 가져오기
  const bucket_list = useSelector(state => state.bucket.list);
```

```
  console.log(bucket_list);

  return (
    <ListStyle>
      {bucket_list.map((list, index) => {
        return (
          <ItemStyle
            className="list_item"
            key={index}
            onClick={() => {
              // 배열의 몇번째 항목을 눌렀는 지, url 파라미터로 넘겨줍니다.
              props.history.push("/detail/"+index);
            }}
          >
            {list}
          </ItemStyle>
        );
      })}
    </ListStyle>
  );
};

const ListStyle = styled.div`
  display: flex;
  flex-direction: column;
  height: 100%;
  overflow-x: hidden;
  overflow-y: auto;
`;

const ItemStyle = styled.div`
  padding: 16px;
  margin: 8px;
  background-color: aliceblue;
`;

export default BucketList;
```

▼ [코드스니펫] - App.js

```
import React from "react";

import { withRouter } from "react-router";
import { Route, Switch } from "react-router-dom";

// import [컴포넌트 명] from [컴포넌트가 있는 파일경로];
import BucketList from "./BucketList";
import styled from "styled-components";
import Detail from "./Detail";
import NotFound from "./NotFound";

// 리덕스 스토어와 연결하기 위해 connect라는 친구를 호출할게요!
import {connect} from 'react-redux';
// 리덕스 모듈에서 (bucket 모듈에서) 액션 생성 함수 두개를 가져올게요!
import {loadBucket, createBucket} from './redux/modules/bucket';

// 이 함수는 스토어가 가진 상태값을 props로 받아오기 위한 함수예요.
const mapStateTopProps = (state) => ({
  bucket_list: state.bucket.list,
});

// 이 함수는 값을 변화시키기 위한 액션 생성 함수를 props로 받아오기 위한 함수예요.
const mapDispatchToProps = (dispatch) => ({
  load: () => {
    dispatch(loadBucket());
  },
  create: (new_item) => {
    console.log(new_item);
    dispatch(createBucket(new_item));
  }
});

// 클래스형 컴포넌트는 이렇게 생겼습니다!
```

```
class App extends React.Component {
  constructor(props) {
    super(props);
    // App 컴포넌트의 state를 정의해줍니다.
    this.state = {

    };
    // ref는 이렇게 선언합니다!
    this.text = React.createRef();
  }

  componentDidMount() {
    console.log(this.props);
  }

  addBucketList = () => {
    const new_item = this.text.current.value;
    this.props.create(new_item);
  };

  // 랜더 함수 안에 리액트 엘리먼트를 넣어줍니다!
  render() {
    return (
      <div className="App">
        <Container>
          <Title>내 버킷리스트</Title>
          <Line />
          {/* 컴포넌트를 넣어줍니다. */}
          {/* <컴포넌트 명 [props 명]={넘겨줄 것(리스트, 문자열, 숫자, ...)}/> */}
          {/* Route 쓰는 법 2가지를 모두 써봅시다! */}
          <Switch>
            <Route path="/" exact component={BucketList} />
            <Route path="/detail/:index" component={Detail} />
            <Route component={NotFound} />
          </Switch>
        </Container>
        {/* 인풋박스와 추가하기 버튼을 넣어줬어요. */}
        <Input>
          <input type="text" ref={this.text} />
          <button onClick={this.addBucketList}>추가하기</button>
        </Input>
      </div>
    );
  }
}

const Input = styled.div`
  max-width: 350px;
  min-height: 10vh;
  background-color: #fff;
  padding: 16px;
  margin: 20px auto;
  border-radius: 5px;
  border: 1px solid #ddd;
`;

const Container = styled.div`
  max-width: 350px;
  min-height: 60vh;
  background-color: #fff;
  padding: 16px;
  margin: 20px auto;
  border-radius: 5px;
  border: 1px solid #ddd;
`;

const Title = styled.h1`
  color: slateblue;
  text-align: center;
`;

const Line = styled.hr`
  margin: 16px 0px;
  border: 1px dotted #ddd;
`;
// withRouter 적용
```

```
// connect로 묶어줬습니다!
export default connect(mapStateTopProps, mapDispatchToProps)(withRouter(App));
```

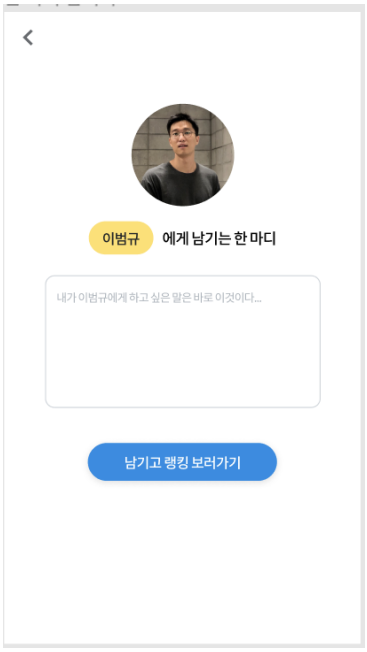
▼ **[코드스니펫] - bucket.js**

```
// widgets.js

// Actions
const LOAD = "bucket/LOAD";
const CREATE = "bucket/CREATE";
const DELETE = "bucket/DELETE";

const initialState = {
  list: ["영화관 가기", "매일 책읽기", "수영 배우기"],
};

// Action Creators
export const loadBucket = (bucket) => {
  return { type: LOAD, bucket };
};

export const createBucket = (bucket) => {
  return { type: CREATE, bucket };
};

export const deleteBucket = (bucket) => {
  return { type: DELETE, bucket };
};

// Reducer
export default function reducer(state = initialState, action) {
    console.log(action);
  switch (action.type) {
    // do reducer stuff
    case "bucket/LOAD":
      return state;

    case "bucket/CREATE":
      console.log(state, action);
      const new_bucket_list = [...state.list, action.bucket];
      return { list: new_bucket_list };

    case "bucket/DELETE":
      const bucket_list = state.list.filter((l, idx) => {
        if(idx !== action.bucket){
            return l;
        }
      });
      return {list: bucket_list};

    default:
      return state;
  }
}
```

12. 끝 & 숙제 설명

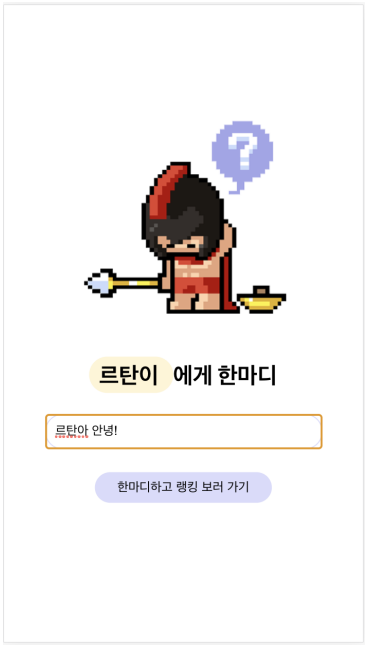
구글 계정 만들어 오기
랭킹화면, 한마디 화면 만들기
각 화면을 라우팅 시키고, 데이터를 리덕스에 넣어주세요.
기능: 첫페이지에서 입력한 이름과, 마지막에 입력하는 한 마디를 리덕스에 넣고, 마지막 랭킹 화면에서 보여줍시다!

▼ 기획서(레이아웃) 보기


이범규
에게 남기는 한 마디
내가 이범규에게 하고 싶은 말은 바로 이것이다...
남기고 랭킹 보러가기


101
명의 사람 중에서 당신은?
78 등
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Blandit quis enim tortor purus.
79 등
이사은
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Blandit quis enim tortor purus.
80 등
이윤석
범규님이 가장 좋아하는 과일이 포도였군요.. 제 사과가 너무 잘 어울리셔서 사과를 제일 좋아하시는 줄 알았어요.
81 등
김서진
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Blandit quis enim tortor purus.
82 등
이형은
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Blandit quis enim tortor purus.
링크 공유하기
재도전하기
83 등
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Blandit quis enim tortor purus.
84 등
신지원
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Blandit quis enim tortor purus.

▼ 예시 화면



13. 3주차 숙제 답안 코드

▼ **[코드스니펫] - 3주차 숙제 답안 코드**

전체 **코드**

▼ **quiz.js**

```
// Actions

// 퀴즈 데이터 가져온다
const GET_QUIZ = "quiz/GET_QUIZ";
// 유저의 응답(퀴즈 답)을 추가한다
const ADD_ANSWER = "quiz/ADD_ANSWER";
// 응답을 초기화 해준다
const RESET_ANSWER = "quiz/RESET_ANSWER";
```

```
const initialState = {
  name: "르탄이",
  score_texts: {
    60: "우린 친구! 앞으로도 더 친하게 지내요! :)",
    80: "우와! 우리는 엄청 가까운 사이!",
    100: "둘도 없는 단짝이에요! :)",
  },
  answers: [],
  quiz: [
    { question: "르탄이는 1살이다.", answer: "O" },
    { question: "르탄이는 2살이다.", answer: "O" },
    { question: "르탄이는 3살이다.", answer: "O" },
    { question: "르탄이는 4살이다.", answer: "O" },
    { question: "르탄이는 5살이다.", answer: "O" },
    // { question: "르탄이는 6살이다.", answer: "O" },
    // { question: "르탄이는 7살이다.", answer: "O" },
    // { question: "르탄이는 8살이다.", answer: "O" },
    // { question: "르탄이는 9살이다.", answer: "O" },
    // { question: "르탄이는 10살이다.", answer: "O" },
    // { question: "르탄이는 11살이다.", answer: "O" },
  ],
};

// Action Creators
export const getQuiz = (quiz_list) => {
  return { type: GET_QUIZ, quiz_list };
};

export const addAnswer = (answer) => {
  return { type: ADD_ANSWER, answer };
};

export const resetAnswer = () => {
  return { type: RESET_ANSWER };
}

// Reducer
export default function reducer(state = initialState, action = {}) {
  switch (action.type) {
    // do reducer stuff
    case "quiz/GET_QUIZ": {
      return { ...state, quiz: action.quiz_list };
    }

    case "quiz/ADD_ANSWER": {
      return { ...state, answers: [...state.answers, action.answer] };
    }

    case "quiz/RESET_ANSWER": {
      return {...state, answers: []};
    }

    default:
      return state;
  }
}
```

▼ **rank.js**

```
// Actions

// 유저 이름을 바꾼다
const ADD_USER_NAME = "rank/ADD_USER_NAME";
// 유저 메시지를 바꾼다
const ADD_USER_MESSAGE = "rank/ADD_USER_MESSAGE";
// 랭킹정보를 추가한다
const ADD_RANK = "rank/ADD_RANK";
// 랭킹정보를 가져온다
const GET_RANK = "rank/GET_RANK";

const initialState = {
  user_name: "",
  user_message: "",
```

```
  user_score: "",
  score_text: {
    60: "우린 친구! 앞으로도 더 친하게 지내요! :)",
    80: "우와! 우리는 엄청 가까운 사이!",
    100: "둘도 없는 단짝이에요! :)",
  },
  ranking: [
    { score: 40, name: "임민영", message: "안녕 르탄아!" },

  ],
};

// Action Creators
export const addUserName = (user_name) => {
  return { type: ADD_USER_NAME, user_name };
};

export const addUserMessage = (user_message) => {
  return { type: ADD_USER_MESSAGE, user_message };
};

export const addRank = (rank_info) => {
  return { type: ADD_RANK, rank_info };
};

export const getRank = (rank_list) => {
  return { type: GET_RANK, rank_list };
};

// Reducer
export default function reducer(state = initialState, action = {}) {
  switch (action.type) {
    // do reducer stuff
    case "rank/ADD_USER_NAME": {
      return { ...state, user_name: action.user_name };
    }

    case "rank/ADD_USER_MESSAGE": {
      return { ...state, user_message: action.user_message };
    }

    case "rank/ADD_RANK": {
      return { ...state, ranking: [...state.ranking, action.rank_info] };
    }

    case "rank/GET_RANK": {
      return { ...state, ranking: action.rank_list };
    }

    default:
      return state;
  }
}
```

▼ configStore.js

```
import { createStore, combineReducers} from "redux";
import quiz from "./modules/quiz";
import rank from "./modules/rank";
import { createBrowserHistory } from "history";

export const history = createBrowserHistory();

const rootReducer = combineReducers({ quiz, rank });
const store = createStore(rootReducer);

export default store;
```

▼ index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { BrowserRouter } from "react-router-dom";

import { Provider } from "react-redux";
import store from "./redux/configStore";

ReactDOM.render(
  <Provider store={store}>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </Provider>,
  document.getElementById("root")
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

▼ App.js

```
import logo from './logo.svg';
import './App.css';
import React from "react";
import {Route, Switch} from "react-router-dom";

import Start from "./Start";
import Quiz from "./Quiz";
import Score from "./Score";
import Message from "./Message";
import Ranking from "./Ranking";

import { withRouter } from "react-router";
// 리덕스 스토어와 연결하기 위해 connect라는 친구를 호출할게요!
import { connect } from "react-redux";


// 이 함수는 스토어가 가진 상태값을 props로 받아오기 위한 함수예요.
const mapStateTopProps = (state) => ({
  ...state,
});

// 이 함수는 값을 변화시키기 위한 액션 생성 함수를 props로 받아오기 위한 함수예요.
const mapDispatchToProps = (dispatch) => ({
  load: () => {

  },

});

class App extends React.Component{
  constructor(props){
    super(props);

    this.state = {

    };
  }

  render () {
    return (
      <div className="App">
        <Switch>
          <Route path="/quiz" component={Quiz} />
          <Route path="/" exact component={Start} />
          <Route path="/score" component={Score} />
```

```
          <Route path="/message" component={Message} />
          <Route path="/ranking" component={Ranking} />
        </Switch>
      </div>
    );
  }
}

export default connect(mapStateTopProps, mapDispatchToProps)(withRouter(App));
```

▼ **Score.js**

```
import React from "react";
import styled from "styled-components";

import { useSelector, useDispatch } from "react-redux";
import {addRank} from "./redux/modules/rank";

const Score = (props) => {
  const name = useSelector((state) => state.quiz.name);
  const score_texts = useSelector((state) => state.quiz.score_texts);

  const answers = useSelector((state) => state.quiz.answers);

  // 정답만 걸러내기
  let correct = answers.filter((answer) => {
    return answer;
  });

  // 점수 계산하기
  let score = (correct.length / answers.length) * 100;

  // 점수별로 텍스트를 띄워줄 준비!
  let score_text = "";

  // Object.keys는 딕셔너리의 키값을 배열로 만들어주는 친구예요!
  Object.keys(score_texts).map((s, idx) => {
    // 첫번째 텍스트 넣어주기
    if (idx === 0) {
      score_text = score_texts[s];
    }
    // 실제 점수와 기준 점수(키로 넣었던 점수) 비교해서 텍스트를 넣자!
    score_text = parseInt(s) <= score ? score_texts[s] : score_text;
  });

  return (
    <ScoreContainer>
      <Text>
        <span>{name}</span>
        퀴즈에 <br />
        대한 내 점수는?
      </Text>
      <MyScore>
        <span>{score}</span>점<p>{score_text}</p>
      </MyScore>

      <Button
        onClick={() => {

          props.history.push("/message");
        }}
        outlined
      >
        {name}에게 한마디
      </Button>
    </ScoreContainer>
  );
};

const ScoreContainer = styled.div`
  display: flex;
  width: 100vw;
  height: 100vh;
  overflow: hidden;
```

```
  padding: 16px;
  box-sizing: border-box;
  flex-direction: column;
  justify-content: center;
  align-items: center;
`;

const Text = styled.h1`
  font-size: 1.5em;
  margin: 0px;
  line-height: 1.4;
  & span {
    background-color: #fef5d4;
    padding: 5px 10px;
    border-radius: 30px;
  }
`;

const MyScore = styled.div`
  & span {
    border-radius: 30px;
    padding: 5px 10px;
    background-color: #fef5d4;
  }
  font-weight: 600;
  font-size: 2em;
  margin: 24px;

  & > p {
    margin: 24px 0px;
    font-size: 16px;
    font-weight: 400;
  }
`;
const Button = styled.button`
  padding: 8px 24px;
  background-color: ${(props) => (props.outlined ? "#ffffff" : "#dadafc")};
  border-radius: 30px;
  margin: 8px;
  border: 1px solid #dadafc;
  width: 80vw;
`;

export default Score;
```

▼ **Quiz.js**

```
import React from "react";
import styled from "styled-components";
import Score from "./Score";
import SwipeItem from "./SwipeItem";


import { useSelector, useDispatch } from "react-redux";
import {addAnswer} from "./redux/modules/quiz";

const Quiz = (props) => {
  const dispatch = useDispatch();
  const answers = useSelector((state) => state.quiz.answers);
  const quiz = useSelector((state) => state.quiz.quiz);

  const num = answers.length;

  const onSwipe = (direction) => {
    let _answer = direction === "left"? "O" : "X";

    if(_answer === quiz[num].answer){
      // 정답일 경우,
      dispatch(addAnswer(true));
    }else{
      // 오답일 경우,
      dispatch(addAnswer(false));
    }
  }
```

```
  if (num > quiz.length -1) {
    return <Score {...props}/>;
    // return <div>퀴즈 끝!</div>;
  }

  return (
    <QuizContainer>
      <p>
        <span>{num + 1}번 문제</span>
      </p>
      {quiz.map((l, idx) => {
        if (num === idx) {
          return <Question key={idx}>{l.question}</Question>;
        }
      })}

      <AnswerZone>
        <Answer>{"O "}</Answer>
        <Answer>{" X"}</Answer>
      </AnswerZone>

      {quiz.map((l, idx) => {
        if (idx === num) {
          return <SwipeItem key={idx} onSwipe={onSwipe}/>;
        }
      })}
    </QuizContainer>
  );
};

const QuizContainer = styled.div`
  margin-top: 16px;
  width: 100%;
  & > p > span {
    padding: 8px 16px;
    background-color: #fef5d4;
    // border-bottom: 3px solid #ffd6aa;
    border-radius: 30px;
  }
`;

const Question = styled.h1`
  font-size: 1.5em;
`;

const AnswerZone = styled.div`
  width: 100%;
  display: flex;
  flex-direction: row;
  min-height: 70vh;
`;

const Answer = styled.div`
  width: 50%;
  display: flex;
  justify-content: center;
  align-items: center;
  font-size: 100px;
  font-weight: 600;
  color: #dadafc77;
`;

const DragItem = styled.div`
  display: flex;
  align-items: center;
  justify-content: center;
  position: fixed;
  top: 0;
  left: 0;
  width: 100vw;
  height: 100vh;

  & > div {
    border-radius: 500px;
    background-color: #ffd6aa;
  }
```

```
  & img {
    max-width: 150px;
  }
`;
export default Quiz;
```

▼ **Start.js**

```
import React from "react";
import img from "./scc_img01.png";
import { useDispatch, useSelector } from "react-redux";
import { addUserName } from "./redux/modules/rank";

const Start = (props) => {
  const dispatch = useDispatch();
  const name = useSelector((state) => state.quiz.name);
  const input_text = React.useRef(null);

  // 컬러셋 참고: https://www.shutterstock.com/ko/blog/pastel-color-palettes-rococo-trend/
  return (
    <div
      style={{
        display: "flex",
        height: "100vh",
        width: "100vw",
        overflow: "hidden",
        padding: "16px",
        boxSizing: "border-box",
      }}
    >
      <div
        className="outter"
        style={{
          display: "flex",
          alignItems: "center",
          justifyContent: "center",
          flexDirection: "column",
          height: "100vh",
          width: "100vw",
          overflow: "hidden",
          padding: "0px 10vw",
          boxSizing: "border-box",
          maxWidth: "400px",
          margin: "0px auto",
        }}
      >
        <img
          src={img}
          style={{ width: "80%", margin: "-70px 16px 48px 16px" }}
        />
        <h1 style={{ fontSize: "1.5em", margin: "0px", lineHeight: "1.4" }}>
          나는{" "}
          <span
            style={{
              backgroundColor: "#fef5d4",
              padding: "5px 10px",
              borderRadius: "30px",
            }}
          >
            {name}
          </span>
          에 대해 얼마나 알고 있을까?
        </h1>
        <input
          ref={input_text}
          type="text"
          style={{
            padding: "10px",
            margin: "24px 0px",
            border: "1px solid #dadafc",
            borderRadius: "30px",
            width: "100%",
            // backgroundColor: "#dadafc55",
          }}
```

```
          placeholder="내 이름"
        />
        <button
          onClick={() => {
            // 이름 저장
            dispatch(addUserName(input_text.current.value));
            // 페이지 이동
            props.history.push("/quiz");
          }}
          style={{
            padding: "8px 24px",
            backgroundColor: "#dadafc",
            borderRadius: "30px",
            border: "#dadafc",
          }}
        >
          시작하기
        </button>
      </div>
    </div>
  );
};

export default Start;
```

▼ **Ranking.js**

```
import React from "react";
import styled from "styled-components";

import { useSelector, useDispatch } from "react-redux";
import {resetAnswer} from "./redux/modules/quiz";

const Ranking = (props) => {
  const dispatch = useDispatch();
  const _ranking = useSelector((state) => state.rank.ranking);
// Array 내장 함수 sort로 정렬하자!
  // https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort

  const ranking = _ranking.sort((a, b) => {
    // 높은 수가 맨 앞으로 오도록!
    return b.score - a.score;
  });

  return (
    <RankContainer>
      <Topbar>
        <p>
          <span>{ranking.length}명</span>의 사람들 중 당신은?
        </p>
      </Topbar>

      <RankWrap>
        {ranking.map((r, idx) => {
          return (
            <RankItem key={idx} highlight={r.current ? true : false}>
              <RankNum>{idx + 1}등</RankNum>
              <RankUser>
                <p>
                  <b>{r.name}</b>
                </p>
                <p>{r.message}</p>
              </RankUser>
            </RankItem>
          );
        })}
      </RankWrap>

      <Button
        onClick={() => {
          dispatch(resetAnswer());
          window.location.href = '/';
        }}
      >
```

```
        다시 하기
      </Button>
    </RankContainer>
  );
};

const RankContainer = styled.div`
  width: 100%;
  padding-bottom: 100px;
`;

const Topbar = styled.div`
  position: fixed;
  top:0;
  left: 0;
  width: 100vw;
  min-height: 50px;
  border-bottom: 1px solid #ddd;
  background-color: #fff;
  & > p {
    text-align: center;
  }

  & > p > span {
    border-radius: 30px;
    background-color: #fef5d4;
    font-weight: 600;
    padding: 4px 8px;
  }
`;

const RankWrap = styled.div`
  display: flex;
  flex-direction: column;
  width: 100%;
  margin-top: 58px;
`;

const RankItem = styled.div`
  width: 80vw;
  margin: 8px auto;
  display: flex;
  border-radius: 5px;
  border: 1px solid #ddd;
  padding: 8px 16px;
  align-items: center;
  background-color: ${props => props.highlight? "#ffd6aa" : "#ffffff"}
`;

const RankNum = styled.div`
  text-align: center;
  font-size: 2em;
  font-weight: 600;
  padding: 0px 16px 0px 0px;
  border-right: 1px solid #ddd;
`;

const RankUser = styled.div`
  padding: 8px 16px;
  text-align: left;
  & > p{
    &:first-child > b{
      border-bottom: 2px solid #212121;
    }
    margin: 0px 0px 8px 0px;
  }
`;

const Button = styled.button`
position: fixed;
bottom: 5vh;
left: 0;
  padding: 8px 24px;
  background-color: ${(props) => (props.outlined ? "#ffffff" : "#dadafc")};
  border-radius: 30px;
  margin: 0px 10vw;
  border: 1px solid #dadafc;
```

```
  width: 80vw;
`;

export default Ranking;
```

▼ **Message.js**

```
import React from "react";
import img from "./scc_img01.png";
import { useDispatch, useSelector } from "react-redux";
import {addRank} from "./redux/modules/rank";

const Message = (props) => {
  const dispatch = useDispatch();
  const name = useSelector((state) => state.quiz.name);
  const answers = useSelector((state) => state.quiz.answers);
  const user_name = useSelector((state)=>state.rank.user_name);

  const input_text = React.useRef(null);
  // 정답만 걸러내기
  let correct = answers.filter((answer) => {
    return answer;
  });

  // 점수 계산하기
  let score = (correct.length / answers.length) * 100;

  // 컬러셋 참고: https://www.shutterstock.com/ko/blog/pastel-color-palettes-rococo-trend/
  return (
    <div
      style={{
        display: "flex",
        height: "100vh",
        width: "100vw",
        overflow: "hidden",
        padding: "16px",
        boxSizing: "border-box",
      }}
    >
      <div
        className="outter"
        style={{
          display: "flex",
          alignItems: "center",
          justifyContent: "center",
          flexDirection: "column",
          height: "100vh",
          width: "100vw",
          overflow: "hidden",
          padding: "0px 10vw",
          boxSizing: "border-box",
          maxWidth: "400px",
          margin: "0px auto",
        }}
      >
        <img
          src={img}
          style={{ width: "80%", margin: "-70px 16px 48px 16px" }}
        />
        <h1 style={{ fontSize: "1.5em", margin: "0px", lineHeight: "1.4" }}>
          <span
            style={{
              backgroundColor: "#fef5d4",
              padding: "5px 10px",
              borderRadius: "30px",
            }}
          >
            {name}
          </span>
          에게 한마디
        </h1>
        <input
        ref={input_text}
          type="text"
```

```
          style={{
            padding: "10px",
            margin: "24px 0px",
            border: "1px solid #dadafc",
            borderRadius: "30px",
            width: "100%",
          }}
          placeholder="한 마디 적기"
        />
        <button
          onClick={() => {

            let rank_info = {
                score: parseInt(score),
                name: user_name,
                message: input_text.current.value,
                current: true,
            };
            // 랭킹 정보 넣기
            dispatch(addRank(rank_info));
            // 주소 이동
            props.history.push('/ranking');

          }}
          style={{
            padding: "8px 24px",
            backgroundColor: "#dadafc",
            borderRadius: "30px",
            border: "#dadafc",
          }}
        >
          한마디하고 랭킹 보러 가기
        </button>
      </div>
    </div>
  );
};

export default Message;
```

Copyright ⓒ TeamSparta All rights reserved.