



[스파르타코딩클럽] 리액트 기초반 - 3주차 (1)



매 주차 강의자료 시작에 PDF파일을 올려두었어요!

[수업 목표]

1. event listener를 사용할 수 있다.
2. React-router로 주소에 따라 다른 페이지를 보여줄 수 있다.
3. 미리 정해놓은 주소가 아닐 때, '앗! 잘못 찾아오셨어요!' 페이지를 보여줄 수 있다.
4. Redux로 상태를 관리해 보고 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!)
5. Redux hook을 사용해본다.

[목차]

- 01. Event Listener
- 02. 라우팅이란?
- 03. 리액트에서 라우팅 처리하기 (1)
- 04. 리액트에서 라우팅 처리하기 (2)
- 05. Quiz 버킷리스트 상세 페이지 만들고 이동시키기
- 06. 라우팅, 조금 더 꼼꼼히 살펴보면?
- 07. 리덕스를 통한 리액트 상태관리
- 08. 리덕스 살펴보기
- 09. 리덕스 써보기
- 10. 리덕스와 컴포넌트를 연결하자!
- 11. 컴포넌트에서 리덕스 데이터 사용하기
- 12. Quiz 버킷 리스트 데이터러를 삭제해보기
- 13. 끝 & 숙제 설명
- HW. 3주차 숙제 답안 코드



모든 토글을 열고 닫는 단축키

Windows : **Ctrl** + **alt** + **t**

Mac : **⌘** + **⌥** + **t**

01. Event Listener

▼ 1) 이벤트 리스너란?



이벤트 리스너는 사용자가 어떤 행동(=이벤트)을 하는 지 아닌 지 지켜보다가 알려주는 것입니다.

대표적으로는 마우스 클릭, 터치, 마우스 오버, 키보드 누름 등이 자주 쓰여요!

더 많은 이벤트가 궁금하다면? →



이벤트 리스너는 <div onClick={} >에서처럼 엘리먼트에 직접 넣어줄 수도 있지만, 이번 강의에서는 addEventListener를 통해 추가해볼거예요.

눈으로만 보고 어떻게 쓰는구나 감만 잡아봅시다. 😊

- 새 프로젝트에서 시작해요! 아래에 코드 스니펫을 준비해두었습니다.

▼ [코드스니펫] - 클래스형 컴포넌트 App.js

```
import logo from './logo.svg';
import './App.css';
import React from "react";
import Text from "./Text";

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

    this.state = {};
    this.circle = React.createRef(null);
  }

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

  componentWillUnmount() {

  }

  render() {
    return (
      <div style={{width: "100vw", height:"100vh", textAlign:"center"}}>
        <Text/>
        <div style={{margin:"auto", width: "250px", height: "250px", background:"green", borderRadius:"250px"}} ref={this
        </div>
      </div>
    );
  }
}

export default App;
```

▼ [코드 스니펫] - 함수형 컴포넌트 Text.js

```
import React from "react";

const Text = (props) => {
  const text = React.useRef(null);

  React.useEffect(() => {

  }, []);
  return (
    <h1 ref={text}>텍스트입니다!</h1>
  )
}

export default Text;
```

▼ 2) 클래스형 컴포넌트에서 event listener 구독하기



이벤트 리스너는 어디에 위치해야할까요?

클릭을 하건, 마우스를 올리건 DOM 요소가 있어야 이벤트가 발생하는 지 지켜볼 수 있겠죠?

→ 네! componentDidMount()에 넣어주면 됩니다.

- (1) 어떤 행동(=이벤트 발생!) 뒤에 실행할 함수 먼저 만들어요.

```
//App.js

hoverEvent = (e) => {
```

```
// 콘솔로 이 이벤트가 누구에게서 일어났는 지 확인할 수 있습니다.
console.log(e.target);
// ref랑 같은 녀석인 지 확인해봐요!
console.log(this.circle.current);

this.circle.current.style.background = "yellow";
}

render(){...}
```

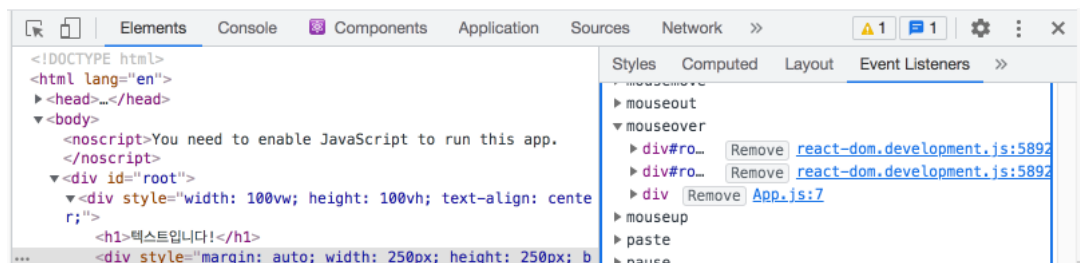
- (2) 이제 `addEventListener()`를 이용해서 이벤트를 등록합니다.

```
//App.js

componentDidMount(){
  // 리액트 요소가 잘 잡혔나 확인해봅시다!
  console.log(this.circle);

  // 마우스를 올렸을 때, 이벤트가 일어나는 지 확인해봅시다.
  this.circle.current.addEventListener("mouseover", this.hoverEvent);
}
```

- 개발자 도구를 통해서도 확인할 수 있습니다 😊



- (3) 이벤트는 꼭 컴포넌트가 사라지면 지워주세요!

👉 이 과정을 **clean up** 이라고 불러요.
 개발자 도구에서 확인했듯, 이벤트는 한번 등록되면 계속 남아있거든요!
 그런데 컴포넌트가 사라지면요? 이벤트가 실행되지는 않겠지만, 남아는 있을거예요. 😞
 그러니 깨끗하게 정돈해주는 과정이 필요합니다! 그게 **clean up**이구요!

```
//App.js

componentWillUnmount() {
  this.circle.current.removeEventListener("mouseover", this.hoverEvent);
}
```

▼ 3) 함수형 컴포넌트에서 event listener 구독하기

👉 **이벤트 리스너는 어디에 위치해야할까요?**
 클릭을 하건, 마우스를 올린건 DOM 요소가 있어야 이벤트가 발생하는 지 지켜볼 수 있겠죠?
 → 그럼 함수형 컴포넌트에서는 `componentDidMount()` 역할을 하는 친구를 가져다 써야겠네요!
`useEffect()` 혹은 써봅시다!

- (1) `useEffect()`

📄 **`useEffect()`는 리액트 훅**이에요.
 라이프 사이클 함수 중 `componentDidMount`와 `componentDidUpdate`, `componentWillUnmount`를 합쳐둔 거라고
 생각하면 금방 이해할 수 있어요!

```
// 첫번째 인자는 익숙하죠! 화살표 함수! 냅, 렌더링 시 실행할 함수가 여기에 들어갑니다.
// 두번째 인자의 []! 디펜던시 어레이라고 불러요. 여기 넣어준 값이 변하면 첫번째 인자인 콜백함수를 실행합니다.
React.useEffect(() => {
  // 여기가 rendering 때 실행될 구문이 들어가는 부분입니다.
  // componentDidMount, componentDidUpdate일 때 동작하는 부분이 여기예요.
  // do something ...

  return () => {
    // 여기가 clean up 부분입니다.
    // componentWillUnmount 때 동작하는 부분이 여기예요.
    //do something ...
  };
}, []);
```

- (2) 어떤 행동(=이벤트 발생!) 뒤에 실행할 함수 먼저 만들어요.

```
//Text.js

...
const hoverEvent = (e) => {
  // 콘솔로 이 이벤트가 누구에게서 일어났는 지 확인할 수 있습니다.
  console.log(e.target);
  // ref랑 같은 녀석인 지 확인해봐요!
  console.log(text.current);

  text.current.style.background = "yellow";
};

return ...
```


- (3) 이제 addEventListener()를 이용해서 이벤트를 등록합니다.

```
//Text.js

// 첫번째 인자는 익숙하죠! 화살표 함수! 냅, 렌더링 시 실행할 함수가 여기에 들어갑니다.
// 두번째 인자의 []! 디펜던시 어레이라고 불러요. 여기 넣어준 값이 변하면 첫번째 인자인 콜백함수를 실행합니다.
React.useEffect(() => {
  // 여기가 rendering 때 실행될 구문이 들어가는 부분입니다.
  // componentDidMount, componentDidUpdate일 때 동작하는 부분이 여기예요.
  text.current.addEventListener("mouseover", hoverEvent);

  return () => {
    // 여기가 clean up 부분입니다.
    // componentWillUnmount 때 동작하는 부분이 여기예요.
    //do something ...
  };
}, [text]);
```

- (4) 이벤트는 꼭 컴포넌트가 사라지면 지워주세요!

 useEffect에서 clean up은 return 구문을 이용해서 합니다!

```
//Text.js

// 첫번째 인자는 익숙하죠! 화살표 함수! 냅, 렌더링 시 실행할 함수가 여기에 들어갑니다.
// 두번째 인자의 []! 디펜던시 어레이라고 불러요. 여기 넣어준 값이 변하면 첫번째 인자인 콜백함수를 실행합니다.
React.useEffect(() => {
  // 여기가 rendering 때 실행될 구문이 들어가는 부분입니다.
  // componentDidMount, componentDidUpdate일 때 동작하는 부분이 여기예요.
  text.current.addEventListener("mouseover", hoverEvent);

  return () => {
    // 여기가 clean up 부분입니다.
    // componentWillUnmount 때 동작하는 부분이 여기예요.
    text.current.removeEventListener("mouseover", hoverEvent);
  };
}, [text]);
```

02. 라우팅이란?

▼ 4) 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 정적 자원을 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 비 효율적이거든요.

(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.

▼ 5) 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.

03. 리액트에서 라우팅 처리하기 (1)

▼ 6) react-router-dom 패키지 설치하기

- 먼저 새프로젝트를 만들어주세요! (⇒ Route_ex 프로젝트 만들기!)



이번 강의부터는 **함수형 컴포넌트**로만 진행할거예요!

- react-router-dom 설치



react-router-dom의 6버전과 5버전은 사용 방법이 많이 달라요! 설치할 때 꼭! @5.2.1을 추가하시어 5버전으로 설치 해주세요. (강의는 5버전을 기준으로 합니다. 😊)

```
yarn add react-router-dom@5.2.1
```

▼ 7) react-router-dom 공식 문서를 보자!



react-router-dom 공식 문서를 호다닥 살펴봅시다. 😊

공식문서는 대부분 비슷하게 생겼으니까, 한 번 살펴보면 앞으로는 혼자서도 잘 읽을 수 있을 거예요. 자 그럼 같이 공식문서 보는 법을 익혀봅시다!

[공식문서 보러가기](#)→

04. 리액트에서 라우팅 처리하기 (2)

▼ 6) 페이지를 전환해보자!

▼ (1) index.js에 BrowserRouter 적용하기

- BrowserRouter 적용하기

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

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);

// 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();
```



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

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

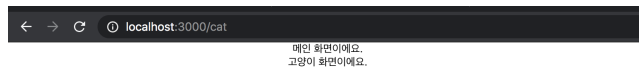
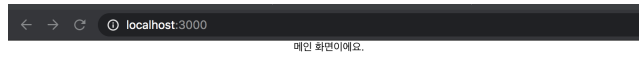
class App extends React.Component {

  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 불러와주세요!
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 파라미터 사용하기

- 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
...
//중복되지 않도록 exact 속성을 써줍니다.
<Route path="/cat" exact component={Cat} />

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

...
```

- 파라미터 사용 방법

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

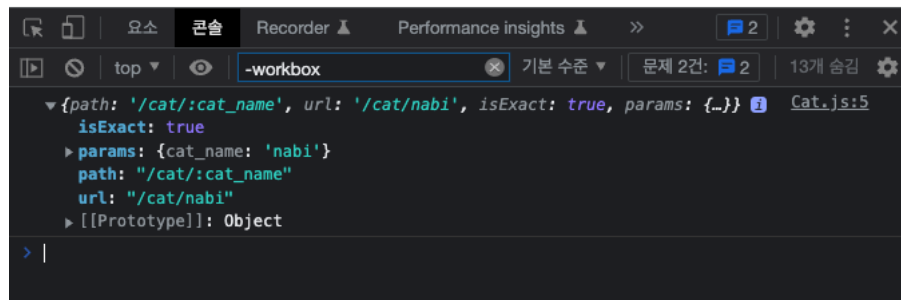
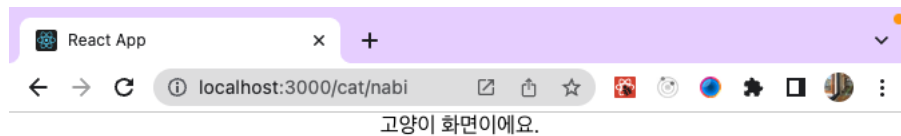
const Cat = (props) => {

  console.log(props.match);

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

export default Cat;
```

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



▼ useParams 혹은 사용해서 이동하기

👉 꼭 props에서 받아오지 않아도, useParams 혹은 사용하면 간단히 파라미터에 접근할 수 있어요! 엄청 편하겠죠. 🤪

```
import React from "react";
import { useParams } from "react-router-dom";
const Cat = (props) => {
  const cat_name = useParams();
  console.log(cat_name);
  // console.log(props);
  return (
    <div>고양이 화면입니다!</div>
  );
};
export default Cat;
```

▼ (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!)

```
// 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";

function App() {
  return (
    <div className="App">
      <div>
        <Link to="/">Home으로 가기</Link>
        <Link to="/cat">Cat으로 가기</Link>
        <Link to="/dog">Dog으로 가기</Link>
      </div>
      { /* 실제로 연결해볼까요! */ }
      <Route path="/" exact>
        <Home />
      </Route>
      <Route path="/cat" component={Cat}>
        { /* <Cat /> */ }
      </Route>
      <Route path="/dog">
        <Dog />
      </Route>
    </div>
  );
}

export default App;
```

▼ -2) history 사용하기

👉 Link 컴포넌트를 클릭하지 않고 페이지를 전환하는 방법 두 가지를 알아봅시다!

▼ props로 history 객체를 받아 이동하기

```
//Dog.js

import React from "react";

const Dog = (props) => {
  // props의 history 객체를 살펴봅시다.
  console.log(props);

  // 그리고 history.push('/home')으로 페이지 이동도 해봐요!

  return (
    <div>
      <div>
        <div>
          onClick={() => {
            props.history.push("/home");
          }}
        </div>
      </div>
      <div>
        강아지 화면이에요.
      </div>
    </div>
  );
};

export default Dog;
```

▼ useHistory 혹은 사용해서 이동하기

👉 꼭 props에서 받아오지 않아도, useHistory 혹은 사용하면 간단히 history 객/체에 접근할 수 있어요! 페이지 이동할 때 써먹으면 엄청 편하겠죠. 🥳

```
//Home.js

import React from "react";
import { useHistory } from "react-router-dom";

const Home = (props) => {
  let history = useHistory();
  return (
    <>
      <div>메인 화면이에요.</div>

      <button
        onClick={() => {
```

```

        history.push("/cat");
      })
    >
    cat으로 가기
  </button>
</>
);
};

export default Home;

```

05. Quiz_버킷리스트 상세 페이지 만들고 이동시키기

- ▼ 7) 🛠 버킷리스트 상세페이지를 만들고 리스트 항목을 누르면 이동시키자!



버킷리스트 프로젝트 안에서도 react-router-dom을 설치해야합니다!

여기에서 **App.js**는 **함수형 컴포넌트**로 바꾸어 해볼게요.

제가 코드 스니펫을 준비했으니, 코드 스니펫으로 복사 → 붙여넣기 하신 후 퀴즈를 진행해주세요!

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



버킷리스트 항목이 나오는 부분만 route로 이동시켜주세요 😊

- ▼ 모습 보기(메인페이지)

내 버킷리스트

영화관 가기

매일 책읽기

수영 배우기

- ▼ 모습 보기(상세페이지)



▼ [코드스니펫] - App.js

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

function App() {

  const [list, setList] = React.useState(["영화관 가기", "매일 책임기", "수영 배우기"]);
  const text = React.useRef(null);

  const addBucketList = () => {
    // 스프레드 문법! 기억하고 계신가요? :)
    // 원본 배열 list에 새로운 요소를 추가해주었습니다.
    setList([...list, text.current.value]);
  }

  return (
    <div className="App">
      <Container>
        <Title>내 버킷리스트</Title>
        <Line />
        {/* 컴포넌트를 넣어줍니다. */}
        {/* <컴포넌트 명 [props 명]={넘겨줄 것(리스트, 문자열, 숫자, ...)}> */}
        <BucketList list={list} />
      </Container>
      {/* 인풋박스와 추가하기 버튼을 넣어줬어요. */}
      <Input>
        <input type="text" ref={text} />
        <button onClick={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. history.{}를 써봐요! ({}엔 뭐가 들어갈까요?)
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(완성)

```

import React from "react";
import styled from "styled-components";
import { Route } from "react-router-dom";

// BucketList 컴포넌트를 import 해옵니다.

```

```
// import [컴포넌트 명] from [컴포넌트가 있는 파일경로];
import BucketList from "../BucketList";
import Detail from "../Detail";

function App() {

  const [list, setList] = React.useState(["영화관 가기", "매일 책읽기", "수영 배우기"]);
  const text = React.useRef(null);

  const addBucketList = () => {
    // 스프레드 문법! 기억하고 계신가요? :)
    // 원본 배열 list에 새로운 요소를 추가해주었습니다.
    setList([...list, text.current.value]);
  }

  console.log(list);
  return (
    <div className="App">
      <Container>
        <Title>내 버킷리스트</Title>
        <Line />
        { /* 컴포넌트를 넣어줍니다. */ }
        { /* <컴포넌트 명 [props 명]={넘겨줄 것(리스트, 문자열, 숫자, ...)}> */ }
        <Route
          path="/"
          exact
          render={(props) => (
            <BucketList list={list}/>
          )}
        />
        <Route path="/detail" component={Detail} />
      </Container>
      { /* 인풋박스와 추가하기 버튼을 넣어줬어요. */ }
      <Input>
        <input type="text" ref={text} />
        <button onClick={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] - Detail.js(완성)

```
// 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";

import { useHistory } from "react-router-dom";

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

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

▼ [코드스니펫] - index.js(완성)

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

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);

// 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();
```


06. 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";
...

return (
  <div className="App">
    ...
    <Switch>
      <Route
        path="/"
        exact
        render={(props) => (
          <BucketList
            list={list}
          />
        )}
      />
      <Route path="/detail" component={Detail} />
    </Switch>
    ...
  </div>
);

...
```

- NotFound컴포넌트를 Route에 주소 없이 연결하면 끝!

```
...
<Switch>
  <Route
    path="/"
    exact
    render={(props) => (
      <BucketList
        list={list}
      />
    )}
  />
  <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!

- NotFound.js에서 useHistory를 가져오는 게 먼저!

```
import { useHistory } from "react-router-dom";
```

- 버튼을 만들어주고,

```
import React from "react";
import { useHistory } from "react-router-dom";

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

export default NotFound;
```

- useHistory를 사용해서 뒤로가기를 만들어요!

```
import React from "react";
import { useHistory } from "react-router-dom";

const NotFound = (props) => {
  let history = useHistory();
  return (
    <div>
      <h1>주소가 올바르지 않아요!</h1>
      <button
        onClick={() => {
          history.goBack();
        }}
      >
        뒤로가기
      </button>
    </div>
  );
};

export default NotFound;
```

07. 리덕스를 통한 리액트 상태관리

- ▼ 10) 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. 😊

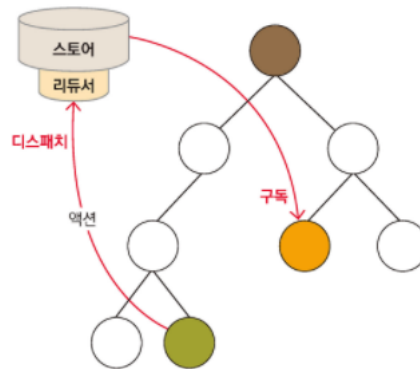
▼ 11) 상태관리 흐름을 알아보자!



[상태관리 흐름도]

딱 4가지만 알면 됩니다! Store, Action, Reducer, 그리고 Component!

아주 큰 흐름만 잘 파악해도 굳곤!



- (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 살펴보기

▼ 12) 리덕스 패키지 설치 & 공식문서 보기



리덕스는 아주 흔히 사용하는 **상태관리 라이브러리**입니다.
전역 상태관리를 편히 할 수 있게 해주는 고마운 친구죠!
 (버킷리스트 프로젝트에서 이어갈거예요!)

- 리덕스 패키지 설치하기

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

- 공식문서 보러가기 →

▼ 13) 리덕스 개념과 용어



리덕스는 데이터를 한 군데 모아놓고, 여기저기에서 꺼내볼 수 있게 해주는 친구입니다.

아래 용어들은 리덕스의 기본 용어인데, 여러분이 키워드 삼기 좋은 용어들이예요. 앞으로 자주 볼 단어들이니 미리 친해집시다!

▼ (1) State



리덕스에서는 저장하고 있는 상태값("데이터"라고 생각하셔도 돼요!)를 **state**라고 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.

▼ 14) 리덕스의 3가지 특징

👉 눈으로 꼭 읽어보세요! 당장 이해하지 못해도 괜찮아요. 프로젝트를 끝낼 쯤엔 이게 그런 소리였구나 하실겁니다. 😎

▼ (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() 혹은 써서만 변경 가능했죠!
데이터가 마구잡이로 변하지 않도록 **불변성**을 유지해 주기 위함입니다.
불변성 뭐냐구요? 간단해요! 허락없이 데이터가 바뀌면 안된단 소리입니다!

조금 더 그럴 듯하게 말하면, 리덕스에 저장된 데이터 = 상태 = state는 읽기 전용입니다.

그런데... 액션으로 변경을 일으킨다면요? 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 한다는 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.

09. 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. 😊 헛갈리실 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 };
}

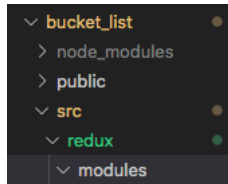
// side effects, only as applicable
// e.g. thunks, epics, etc
export function getWidget () {
  return dispatch => get('/widget').then(widget => dispatch(updateWidget(widget)))
}
```

▼ 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.

```
// 액션 타입을 정해줍니다.  
const LOAD = "bucket/LOAD";  
const CREATE = "bucket/CREATE";
```

▼ (2) initialState



초기 상태값을 만들어줄거예요! 그러니까, 기본 값이죠.

```
// 초기 상태값을 만들어줍니다.  
const initialState = {  
  list: ["영화관 가기", "매일 책읽기", "수영 배우기"],  
};
```

▼ (3) Action Creator



액션 생성 함수를 작성합니다.

```
// 액션 생성 함수예요.  
// 액션을 만들어줄 함수죠!  
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.)

```
// 리듀서예요.
// 실질적으로 store에 들어가 있는 데이터를 변경하는 곳이지!
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";

// 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;
```

10. 리덕스와 컴포넌트를 연결하자!

▼ 17) Store 연결하기

👉 store를 다 만들었으니 이제 컴포넌트와 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/client";
import "../index.css";
import App from "../App";
import reportWebVitals from "../reportWebVitals";
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";

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

// 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();
```

11. 컴포넌트에서 리덕스 데이터 사용하기

▼ 18) 컴포넌트에서 리덕스 액션 사용하는 법

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

▼ (1) 컴포넌트에서 리덕스 데이터 사용하기

▼ -1) 리덕스 훅

👉 리덕스도 훅이 있어요!
상태, 즉, 데이터를 가져오는 것 하나, 상태를 업데이트할 수 있는 것 하나 😊
이렇게 두 가지를 정말 많이 쓴답니다!
더 많은 훅이 궁금하다면? ([훅 보러가기](#)↗)

```
// useDispatch는 데이터를 업데이트할 때,  
// useSelector는 데이터를 가져올 때 씁니다.  
import {useDispatch, useSelector} from "react-redux";
```

▼ -2) BucketList.js에서 redux 데이터 가져오기

👉 `useSelector((state) => state.bucket)`
configStore.js에서 루트 리듀서를 만들었던 거 기억하시나요?
앗, 바로 감이 오셨나요? 네, 맞아요! 여기에서 **state**는 리덕스 스토어가 가진 전체 데이터예요.

```
...  
// redux 훅 중, useSelector를 가져옵니다.  
import { useSelector } from "react-redux";  
  
const BucketList = (props) => {  
  let history = useHistory();  
  // 이 부분은 주석처리!  
  // console.log(props);  
  // const my_lists = props.list;  
  // 여기에서 state는 리덕스 스토어가 가진 전체 데이터예요.  
  // 우리는 그 중, bucket 안에 들어있는 list를 가져옵니다.  
  const my_lists = useSelector((state) => state.bucket.list);  
  return (  
    <ListStyle>  
      {my_lists.map((list, index) => {  
        return (  
          <ItemStyle  
            className="list_item"  
            key={index}  
            onClick={() => {  
              history.push("/detail");  
            }}  
          >  
            {list}  
          </ItemStyle>  
        );  
      })}  
    </ListStyle>  
  );  
};  
...
```

▼ -3) App.js에서 redux 데이터 추가하기



`useSelector((state) => state.bucket)`

configStore.js에서 루트 리듀서를 만들었던 거 기억하시나요?

앗, 바로 감이 오셨나요? 네, 맞아요! 여기에서 **state**는 리덕스 스토어가 가진 전체 데이터예요.

- import 부터!

```
//App.js

// useDispatch를 가져와요!
import {useDispatch} from "react-redux";
// 액션생성함수도 가져오고요!
import { createBucket } from "../redux/modules/bucket";
```

- useDispatch 혹 쓰기

```
const dispatch = useDispatch();

const addBucketList = () => {
  // 스프레드 문법! 기억하고 계신가요? :)
  // 원본 배열 list에 새로운 요소를 추가해주었습니다.
  // 여긴 이제 주석처리!
  // setList([...list, text.current.value]);

  dispatch(createBucket(text.current.value));
};
```

▼ [코드스니펫] - App.js

```
import React from "react";
import styled from "styled-components";
import { Route, Switch } from "react-router-dom";
// useDispatch를 가져와요!
import {useDispatch} from "react-redux";
// 액션생성함수도 가져오고요!
import { createBucket } from "../redux/modules/bucket";

// BucketList 컴포넌트를 import 해옵니다.
// import [컴포넌트 명] from [컴포넌트가 있는 파일경로];
import BucketList from "../BucketList";
import Detail from "../Detail";
import NotFound from "../NotFound";

function App() {

  const text = React.useRef(null);
  // useHistory 사용하는 것과 비슷하죠? :)
  const dispatch = useDispatch();

  const addBucketList = () => {
    // 스프레드 문법! 기억하고 계신가요? :)
    // 원본 배열 list에 새로운 요소를 추가해주었습니다.
    // 여긴 이제 주석처리!
    // setList([...list, text.current.value]);

    dispatch(createBucket(text.current.value));
  };

  return (
    <div className="App">
      <Container>
        <Title>내 버킷리스트</Title>
        <Line />
        { /* 컴포넌트를 넣어줍니다. */ }
        { /* <컴포넌트 명 [props 명]={넘겨줄 것(리스트, 문자열, 숫자, ...)}> */ }
        <Switch>
          { /* <Route
              path="/"
              exact
              render={ (props) => <BucketList list={list} /> }
            */ }
          { /* 이제는 render를 사용해서 list를 넘겨줄 필요가 없죠! 버킷리스트가 리덕스에서 데이터를 알아서 가져갈거니까요! */ }
          <Route exact path="/" component={BucketList} />
          <Route exact path="/detail" component={Detail} />
          <Route component={NotFound} />
        </Switch>
      </Container>
    </div>
  );
}
```

```

    /* 인풋박스과 추가하기 버튼을 넣어줬어요. */
    <Input>
      <input type="text" ref={text} />
      <button onClick={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";

import { useHistory } from "react-router-dom";
// redux 혹은, useSelector를 가져옵니다.
import { useSelector } from "react-redux";

const BucketList = (props) => {
  let history = useHistory();
  // 이 부분은 주석처리!
  // console.log(props);
  // const my_lists = props.list;
  // 여기에서 state는 리덕스 스토어가 가진 전체 데이터예요.
  // 우리는 그 중, bucket 안에 들어있는 list를 가져옵니다.
  const my_lists = useSelector((state) => state.bucket.list);
  return (
    <ListStyle>
      {my_lists.map((list, index) => {
        return (
          <ItemStyle
            className="list_item"
            key={index}
            onClick={() => {
              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) 상세페이지에서 버킷리스트 내용을 띄워보기

▼ -1) 몇 번째 상세에 와있는 지 알기 위해, URL 파라미터를 적용하자

```
// App.js
<Route exact path="/detail/:index" component={Detail} />
```

```
//BucketList.js
...
{my_lists.map((list, index) => {
  return (
    <ItemStyle
      className="list_item"
      key={index}
      onClick={() => {
        history.push("/detail/"+index);
      }}
    >
      {list}
    </ItemStyle>
  );
})}
...
```

▼ -2) 상세페이지에서 버킷리스트 내용을 띄워보자

```
//Detail.js
// 리액트 패키지를 불러옵니다.
import React from "react";
// 라우터 혹은 훅을 불러옵니다.
import {useParams} from "react-router-dom";
// redux hook을 불러옵니다.
import { useSelector } from "react-redux";

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

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

export default Detail;
```

12. 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

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

```
import {useParams} from "react-router-dom";
// redux hook을 불러옵니다.
import { useSelector } from "react-redux";

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

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

export default Detail;
```

▼ [코드스니펫] - BucketList.js

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

import { useHistory } from "react-router-dom";
// redux 쪽 중, useSelector를 가져옵니다.
import { useSelector } from "react-redux";

const BucketList = (props) => {
  let history = useHistory();
  // 이 부분은 주석처리!
  // console.log(props);
  // const my_lists = props.list;
  // 여기에서 state는 리덕스 스토어가 가진 전체 데이터예요.
  // 우리는 그 중, bucket 안에 들어있는 list를 가져옵니다.
  const my_lists = useSelector((state) => state.bucket.list);
  return (
    <ListStyle>
      {my_lists.map((list, index) => {
        return (
          <ItemStyle
            className="list_item"
            key={index}
            onClick={() => {
              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 styled from "styled-components";
import { Route, Switch } from "react-router-dom";
// useDispatch를 가져와요!
import { useDispatch } from "react-redux";
// 액션생성함수도 가져오고요!
import { createBucket } from "../redux/modules/bucket";

// BucketList 컴포넌트를 import 해옵니다.
// import [컴포넌트명] from [컴포넌트가 있는 파일경로];
```

```

import BucketList from "./BucketList";
import Detail from "./Detail";
import NotFound from "./NotFound";

function App() {

  const text = React.useRef(null);
  // useHistory 사용하는 것과 비슷하죠? :)
  const dispatch = useDispatch();

  const addBucketList = () => {
    // 스프레드 문법! 기억하고 계신가요? :)
    // 원본 배열 list에 새로운 요소를 추가해주었습니다.
    // 여긴 이제 주석처리!
    // setList([...list, text.current.value]);

    dispatch(createBucket(text.current.value));
  };

  return (
    <div className="App">
      <Container>
        <Title>내 버킷리스트</Title>
        <Line />
        { /* 컴포넌트를 넣어줍니다. */ }
        { /* <컴포넌트 명 [props 명]={넘겨줄 것(리스트, 문자열, 숫자, ...)}> */ }
        <Switch>
          { /* <Route
              path="/"
              exact
              render={ (props) => <BucketList list={list} /> }
            /> */ }
          { /* 이제는 render를 사용해서 list를 넘겨줄 필요가 없죠! 버킷리스트가 리덕스에서 데이터를 알아서 가져갈거니까요! */ }
          <Route exact path="/" component={BucketList} />
          <Route exact path="/detail/:index" component={Detail} />
          <Route component={NotFound} />
        </Switch>
      </Container>
      { /* 인풋박스과 추가하기 버튼을 넣어줬어요. */ }
      <Input>
        <input type="text" ref={text} />
        <button onClick={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] - redux/modules/bucket.js

```

// 액션 타입을 정해줍니다.
const LOAD = "bucket/LOAD";
const CREATE = "bucket/CREATE";

```

```
// 초기 상태값을 만들어줍니다.
const initialState = {
  list: ["영화관 가기", "매일 책읽기", "수영 배우기"],
};

// 액션 생성 함수예요.
// 액션을 만들어줄 함수죠!
export const loadBucket = (bucket) => {
  return { type: LOAD, bucket };
};

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

// 리듀서예요.
// 실질적으로 store에 들어가 있는 데이터를 변경하는 곳이죠!
export default function reducer(state = initialState, action = {}) {
  switch (action.type) {
    case "bucket/LOAD":
      return state;

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

    default:
      return state;
  }
}
```



힌트:

1. 조건에 맞춰 배열 항목을 필터링 해주는 array의 내장함수 `filter`를 사용합니다.
(`filter`가 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?
→ `create`를 참고해서 `delete`를 만들어보세요!
3. Detail.js에 제가 `useDispatch()` 사용법을 주석처리 해두었습니다 :) 주석을 참고해보세요!

▼ A. 함께하기(완성본)

▼ [코드스니펫] - Detail.js(완성)

```
//Detail.js
// 리액트 페이지를 불러옵니다.
import React from "react";
import { useHistory } from "react-router-dom";
// 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();
  const history = useHistory();

  // 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.
          console.log("삭제하기 버튼을 눌렀어!");
          dispatch(deleteBucket(bucket_index));
          history.goBack();
        }}
      >
        삭제하기
      </button>
    </div>
  );
};
```

```
};

export default Detail;
```

▼ [코드스니펫] - BucketList.js(완성)

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

import { useHistory } from "react-router-dom";
// redux 혹은 중, useSelector를 가져옵니다.
import { useSelector } from "react-redux";

const BucketList = (props) => {
  let history = useHistory();
  // 이 부분은 주석처리!
  // console.log(props);
  // const my_lists = props.list;
  // 여기에서 state는 리덕스 스토어가 가진 전체 데이터예요.
  // 우리는 그 중, bucket 안에 들어있는 list를 가져옵니다.
  const my_lists = useSelector((state) => state.bucket.list);
  return (
    <ListStyle>
      {my_lists.map((list, index) => {
        return (
          <ItemStyle
            className="list_item"
            key={index}
            onClick={() => {
              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(완성)

```
import React from "react";
import styled from "styled-components";
import { Route, Switch } from "react-router-dom";
// useDispatch를 가져와요!
import { useDispatch } from "react-redux";
// 액션생성함수도 가져오고요!
import { createBucket } from "../redux/modules/bucket";

// BucketList 컴포넌트를 import 해옵니다.
// import [컴포넌트명] from [컴포넌트가 있는 파일경로];
import BucketList from "../BucketList";
import Detail from "../Detail";
import NotFound from "../NotFound";

function App() {

  const text = React.useRef(null);
  // useHistory 사용하는 것과 비슷하죠? :)
  const dispatch = useDispatch();

  const addBucketList = () => {
    // 스프레드 문법! 기억하고 계신가요? :)
  }
}
```



```

    // 원본 배열 list에 새로운 요소를 추가해주었습니다.
    // 여긴 이제 주석처리!
    // setList([...list, text.current.value]);

    dispatch(createBucket(text.current.value));
  };

  return (
    <div className="App">
      <Container>
        <Title>내 버킷리스트</Title>
        <Line />
        { /* 컴포넌트를 넣어줍니다. */ }
        { /* <컴포넌트 명 [props 명]={넘겨줄 것(리스트, 문자열, 숫자, ...)}> */ }
        <Switch>
          { /* <Route
              path="/"
              exact
              render={(props) => <BucketList list={list} /> }
            /> */ }
          { /* 이제는 render를 사용해서 list를 넘겨줄 필요가 없죠! 버킷리스트가 리덕스에서 데이터를 알아서 가져갈거니까요! */ }
          <Route exact path="/" component={BucketList} />
          <Route exact path="/detail/:index" component={Detail} />
          <Route component={NotFound} />
        </Switch>
      </Container>
      { /* 인풋박스와 추가하기 버튼을 넣어줬어요. */ }
      <Input>
        <input type="text" ref={text} />
        <button onClick={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] - bucket.js(완성)

```

// 액션 타입을 정해줍니다.
const LOAD = "bucket/LOAD";
const CREATE = "bucket/CREATE";
const DELETE = "bucket/DELETE";

// 초기 상태를 만들어줍니다.
const initialState = {
  list: ["영화관 가기", "매일 책읽기", "수영 배우기"],
};

// 액션 생성 함수예요.
// 액션을 만들어줄 함수죠!
export const loadBucket = (bucket) => {
  return { type: LOAD, bucket };
};

```

```

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

export function deleteBucket(bucket_index) {
  console.log("지울 버킷 인덱스", bucket_index);
  return { type: DELETE, bucket_index };
}

// 리뷰서예요.
// 실질적으로 store에 들어가 있는 데이터를 변경하는 곳이죠!
export default function reducer(state = initialState, action = {}) {
  switch (action.type) {
    case "bucket/LOAD":
      return state;

    case "bucket/CREATE": {
      const new_bucket_list = [...state.list, action.bucket];
      return { list: new_bucket_list };
    }
    case "bucket/DELETE": {
      const new_bucket_list = state.list.filter((l, idx) => {
        return parseInt(action.bucket_index) !== idx;
      });
      return { list: new_bucket_list };
    }
    default:
      return state;
  }
}

```

13. 끝 & 숙제 설명



구글 계정 만들어 오기

퀴즈 화면, 점수 화면 만들기

각 화면을 라우팅 시키고, 퀴즈 데이터를 리덕스에 넣어주세요.

기능:

- 첫페이지에서 입력한 이름과 퀴즈 데이터, 점수는 리덕스에 넣어주세요.
- 퀴즈 화면에서 O, X를 누르면 다음 퀴즈를 보여주세요.
- 퀴즈를 다 풀면 점수화면으로 이동해주세요.
- 점수 화면에서는 첫페이지에서 입력한 이름과, 점수를 보여줍니다.



숙제 시작하기 전 꼭꼭 생각해 볼 것!

1. 퀴즈 데이터에는 뭐가 필요할까요? 퀴즈 한 문제가 어떤 데이터로 구성되어야 하는 지 잘 생각해보세요! 😊

→ 퀴즈 하나에 필요한 건 문제와 답! 퀴즈 하나는 {question: "문제예요!", answer: "O"} 이렇게 구성되어야 할 거예요.

2. 그럼 점수를 내기 위해서는 유저의 답안을 어떻게 보관하면 좋을까요?

→ 따로 답만 추려 배열([])을 만들어 보관해도 좋고, 퀴즈 데이터에 추가해도 괜찮아요. (이 경우 퀴즈 하나가 {question: "문제예요!", answer: "O", user_answer: "X"}로 구성되겠죠.)

3. 점수 계산은 언제 해야할까요?

→ 퀴즈를 다 풀고 나서 해야합니다! 다 풀고 점수 계산한 후 점수 페이지로 이동해도 좋고, 다풀고 바로 점수 페이지로 이동한 뒤에 점수 계산을 해도 좋아요. 🤔

▼ 기획서(레이아웃) 보기

점수 보기

이범규 퀴즈에 대한 내 점수는

58 점

우와! 이 정도면 가깝다고도 멀다고도 할 수 없는 그런 애매한 사이군요! 조금 더 관심을 갖고 친해지면 어떨까요?

정답 보기

이범규에게 한 마디

문제풀기



1번째 문제

이범규가 가장 좋아하는 과일은 사과이다.

☐ O ☐ X

▼ 예시 화면

르탄이 퀴즈에
대한 내 점수는?

100 점

둘도 없는 단짝이에요! :)

르탄이에게 한마디

1번 문제

르탄이는 1살이다.

O X

HW. 3주차 숙제 답안 코드

▼ [코드스니펫] - 3주차 숙제 답안 코드

전체 코드

▼ App.js

```
import React from "react";
import './App.css';

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

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

function App() {
  const [name, setName] = React.useState("르탄이");

  return (
    <div
      className="App"
      style={{
        maxWidth: "350px",
        margin: "auto",
      }}
    >
      <Route path="/" exact>
        <Start name={name} />
      </Route>

      <Route path="/quiz" exact>
        <Quiz />
      </Route>

      <Route path="/score" exact>
        <Score name={name} />
      </Route>
    </div>
  );
}

export default App;
```

▼ Score.js

```
import React from "react";
import { useSelector } from "react-redux";
import { Container, Button, Img, Highlight } from "./elements";

const Score = (props) => {
  const quiz_list = useSelector((state) => state.quiz.quiz_list);
  const user_answer_list = useSelector((state) => state.quiz.user_answer_list);

  const _score =
    (100 / quiz_list.length) *
    quiz_list.filter((q, idx) => {
      return q.answer === user_answer_list[idx];
    }).length;

  const score = Math.round(_score);
  return (
    <Container is_main>
      <h3>
        <Highlight>{props.name}</Highlight> 퀴즈에 대한 내 점수는 <br />
        <Highlight>{score}</Highlight>점
      </h3>

      <p>우와! 우린 참 천재요!</p>

      <Button>{props.name}에게 한 마다</Button>
    </Container>
  );
};

export default Score;
```

▼ Quiz.js

```
import React from "react";
import rtan from "../scc_img01.png";

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

import { Container, Button, Img, Highlight } from "../elements";

const Quiz = (props) => {
  const history = useHistory();
  const dispatch = useDispatch();

  const quiz_list = useSelector((state) => state.quiz.quiz_list);
  const user_answer_list = useSelector((state) => state.quiz.user_answer_list);

  const setAnswer = (user_answer) => {
    dispatch(addAnswer(user_answer));
  };

  React.useEffect(() => {
    if (user_answer_list.length === quiz_list.length) {
      history.push("/score");
      return;
    }
  }, [user_answer_list]);

  if (user_answer_list.length === quiz_list.length) {
    return null;
  }

  return (
    <Container>
      <div>
        <p>
          <Highlight>{user_answer_list.length + 1}번 문제</Highlight>
        </p>
        <h3>{quiz_list[user_answer_list.length].question}</h3>
      </div>
      <Img src={rtan} />

      <div>
        <Button
          onClick={() => {
            setAnswer(true);
          }}
          style={{ width: "50px", height: "50px", margin: "16px" }}
        >
          O
        </Button>
        <Button
          onClick={() => {
            setAnswer(false);
          }}
          style={{ width: "50px", height: "50px", margin: "16px" }}
        >
          X
        </Button>
      </div>
    </Container>
  );
};

export default Quiz;
```

▼ Start.js

```
import React from "react";
import img from "../scc_img01.png";
import styled from "styled-components";

import { useHistory } from "react-router-dom";
import { useDispatch } from "react-redux";
import { setName } from "../redux/modules/user";
import { Container, Button, Img, Highlight } from "../elements";

const Start = (props) => {
  const history = useHistory();
  const dispatch = useDispatch();
  const name_ref = React.useRef(null);
```

```

return (
  <Container is_main>
    <Img
      src={img}
      style={{
        width: "60vw",
        margin: "16px",
      }}
    />
    <h1>
      나는{" "}
      <Highlight
        style={{
          backgroundColor: "#fef5d4",
          padding: "5px 10px",
          borderRadius: "30px",
        }}
      >
        {props.name}
      </Highlight>
      에 대해 얼마나 알고 있을까?
    </h1>
    <input
      ref={name_ref}
      style={{
        border: "1px solid #dadafc",
        borderRadius: "30px",
        padding: "10px",
        width: "100%",
      }}
    />
    <Button
      onClick={() => {
        dispatch(setName(name_ref.current.value));
        history.push("/quiz");
      }}
    >
      시작하기
    </Button>
  </Container>
);
};

export default Start;

```

▼ index.js

```

import React from "react";
import ReactDOM from "react-dom/client";
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";

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

// 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();

```

▼ elements.js (styled component 적용 부분입니다 / 필수 제출 x)

```

import styled from "styled-components";

export const Container = styled.div`
  display: flex;
  height: 100vh;
  flex-direction: column;
  align-items: center;
  justify-content: ${props => props.is_main ? "center" : "space-between"};

```

```
padding: 16px;
box-sizing: border-box;
`;

export const Button = styled.button`
padding: 10px 36px;
background: #dadafc;
border: #dadafc;
border-radius: 30px;
margin: 36px 0px;
`;

export const Img = styled.img`
width: 60vw;
margin: 16px;
`;

export const Highlight = styled.span`
font-weight: bold;
background: #fef5d4;
padding: 5px 10px;
border-radius: 30px;
`;
```

▼ redux/configStore.js

```
import { createStore, combineReducers } from "redux";
import quiz from "../modules/quiz";
import user from "../modules/user";

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

export default store;
```

▼ redux/modules/quiz.js

```
// 어떤 데이터를 넣을거야 -> 퀴즈 목록 / 유저 정답 목록
// 어떻게 수정 해볼거야 -> 유저가 선택한 오엑스 정답을 정답 목록에 추가해줄거야!

const ADD_ANSWER = "quiz/ADD_ANSWER";

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

const initialState = {
  quiz_list: [
    { question: "르탕이는 1살이다.", answer: false },
    { question: "르탕이는 2살이다.", answer: false },
    { question: "르탕이는 3살이다.", answer: true },
  ],
  user_answer_list: [],
};

export default function reducer(state = initialState, action = {}) {
  switch (action.type) {
    case "quiz/ADD_ANSWER": {
      console.log(action);
      const new_user_answer_list = [...state.user_answer_list, action.user_answer];

      console.log(new_user_answer_list);
      return { ...state, user_answer_list: new_user_answer_list };
    }

    default:
      return state;
  }
}
```

▼ redux/modules/user.js

```
// user 이름을 날자!
// 이름을 바꿔주자!

const SET_NAME = "user/SET_NAME";

export const setName = (name) => {
```



```
        return {type: SET_NAME, name};
    }

    const initialState = {
        user_name: "",
    }

    export default function reducer(state = initialState, action ={})
    {
        switch (action.type) {
            case "user/SET_NAME": {
                console.log(action);
                return {...state, user_name: action.name};
            }

            default:
                return state;
        }
    }
}
```

Copyright © TeamSparta All rights reserved.