



**Seoul
Software
ACademy**

with





useEffect

실전 활용!



```
import React, { useEffect, useState } from "react";

export default function UseEffectFetch() {
  const [dataArr, setDataArr] = useState([]);

  async function fetchData() {
    const resFetch = await fetch("http://localhost:4000/", {
      method: "GET",
      headers: {
        "Content-type": "application/json",
      },
    });
    if (resFetch.status !== 200) return alert('통신 에러');

    const data = await resFetch.json();
    setDataArr(data);
    console.log(data);
  }

  useEffect(() => {
    fetchData();
  }, []);
}
```

```
▼ (3) [{...}, {...}, {...}] ⓘ
  ▶ 0: {name: '뽀로로', age: '5', nickName: '사고뭉치'}
  ▶ 1: {name: '루피', age: '4', nickName: '공주님'}
  ▶ 2: {name: '크롱이', age: '5', nickName: '장난꾸러기'}
    length: 3
  ▶ [[Prototype]]: Array(0)
```



```
import ProfileComponent from "../ProfileComponent";  
// 기존 코드
```

```
return (  
  <>  
    {dataArr.map((el) => {  
      return (  
        <ProfileComponent  
          key={el.name}  
          name={el.name}  
          age={el.age}  
          nickName={el.nickName}  
        />  
      );  
    })}  
  </>  
);
```



이름 : 뽀로로

나이 : 5

별명 : 사고뭉치

이름 : 루피

나이 : 4

별명 : 공주님

이름 : 크롱이

나이 : 5

별명 : 장난꾸러기

```
Elements Console Sources Network Performance
top Filter
Download the React DevTools for a better development experience
act-devtools
(3) [{...}, {...}, {...}]
  0: {name: '뽀로로', age: '5', nickName: '사고뭉치'}
  1: {name: '루피', age: '4', nickName: '공주님'}
  2: {name: '크롱이', age: '5', nickName: '장난꾸러기'}
  length: 3
  [[Prototype]]: Array(0)
(3) [{...}, {...}, {...}]
>
```



컴포넌트 꾸미기



Styled Components



styled
components



Styled Components 설치 및 불러오기

- 먼저 설치 부터 하시죠!
- Npm install styled-components
- 그리고 TestStyled.jsx 파일 만들기
- Styled 컴포넌트 모듈 불러오기

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



Styled Components 사용하기

- Styled Components 는 독특하게 사용이 됩니다!

```
export default function TestStyled() {  
  return (  
    <MyDiv>  
      <MyHeading>h1 태그 입니다!</MyHeading>  
      <MySpan>span 태그 입니다!</MySpan>  
    </MyDiv>  
  )  
}
```

src/components/TestStyled.jsx

- 먼저 자기만의 이름으로 태그를 구성 합니다!



Styled Components 사용하기

- 각각의 태그를 변수에 할당하고 해당 태그의 실제적인 태그명을 styled 를 이용하여 지정합니다!

```
const MyDiv = styled.div`  
  background-color: orange;  
`;  
;
```

```
const MyHeading = styled.h1`  
  color: blue;  
`;
```

```
const MySpan = styled.span`  
  background-color: pink;  
  font-weight: 700;  
`;
```

src/components/TestStyled.jsx



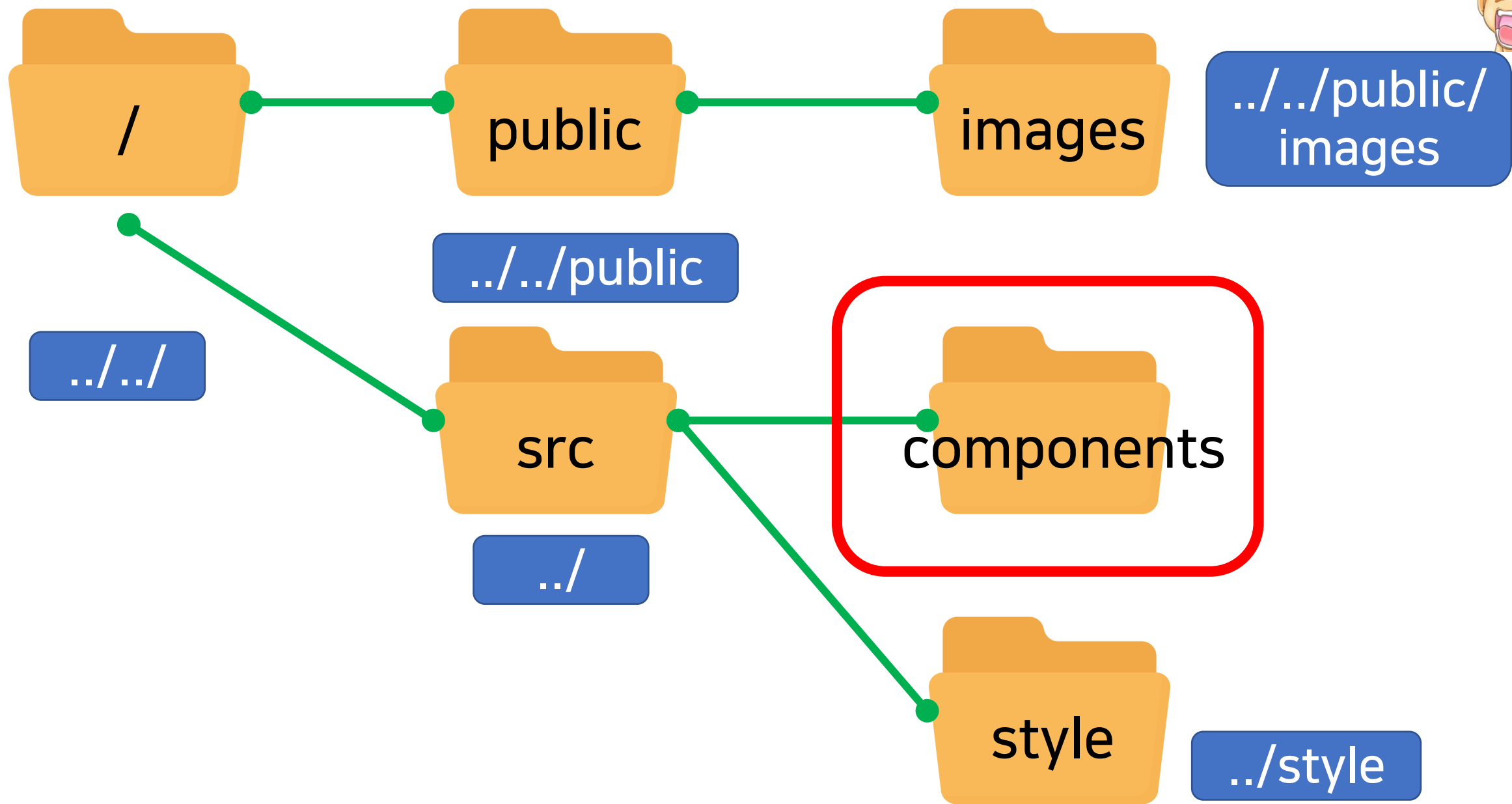
Styled Components 사용하기

- 자신이 지정한 태그의 이름은 일종의 빈 태그로 만들어 지지만 styled 모듈을 사용하여 해당 태그의 실질적 태그를 지정할 수 있습니다
- 그리고 CSS 요소는 뒤에 따라오는 `` 백틱 문자의 내부에 써주시면 됩니다!



Public

폴더 사용







폐북의 가호

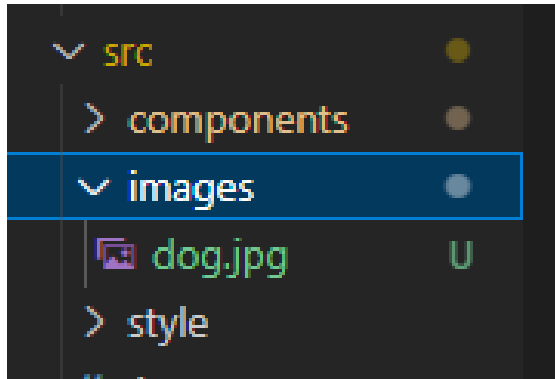
- `Npx create-react-app` 을 통해 만들어진 리액트 앱의 경우는 public 폴더가 자동으로 static 처리가 됩니다!
- 따라서, 어느 위치에서건 `/` 를 써서 접근하면 public 폴더가 호출 됩니다!
- 그럼 강아지 사진의 주소 값을 `/images/dog.jpg` 로 변경해 봅시다!



```
export default function Image() {  
  return (  
    <>  
      
    </>  
  )  
}
```

src/components/Image.js





```
import dogImg from "../images/dog.jpg"

export default function Image() {
  return (
    <>
      <img src={dogImg} alt="강아지" />
    </>
  )
}
```

src/components/Image.js

src 폴더 내부 참조는 문제 없습니다!





Props.children



```
export default function FancyBorder(props) {  
  return (  
    <div style={{ border: `3px solid ${props.color}` }}>  
      {props.children}  
    </div>  
  );  
}
```

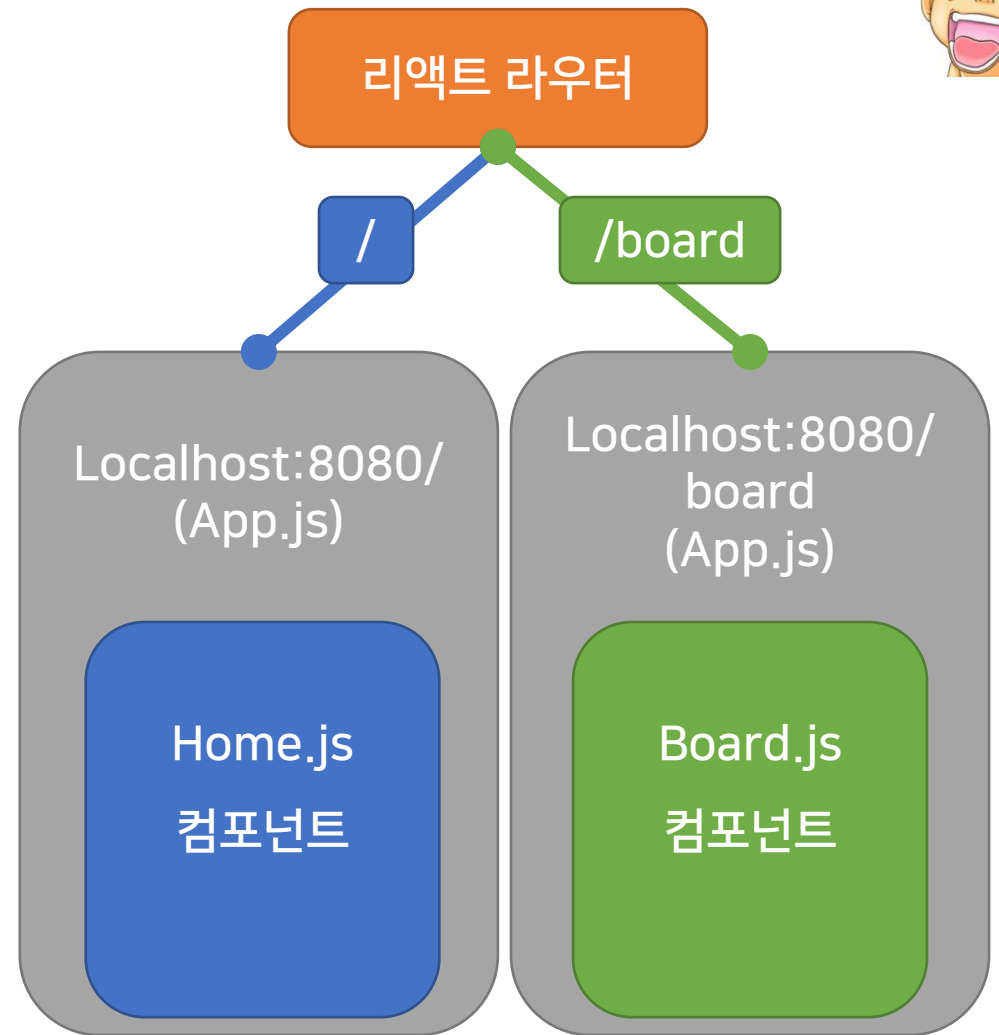
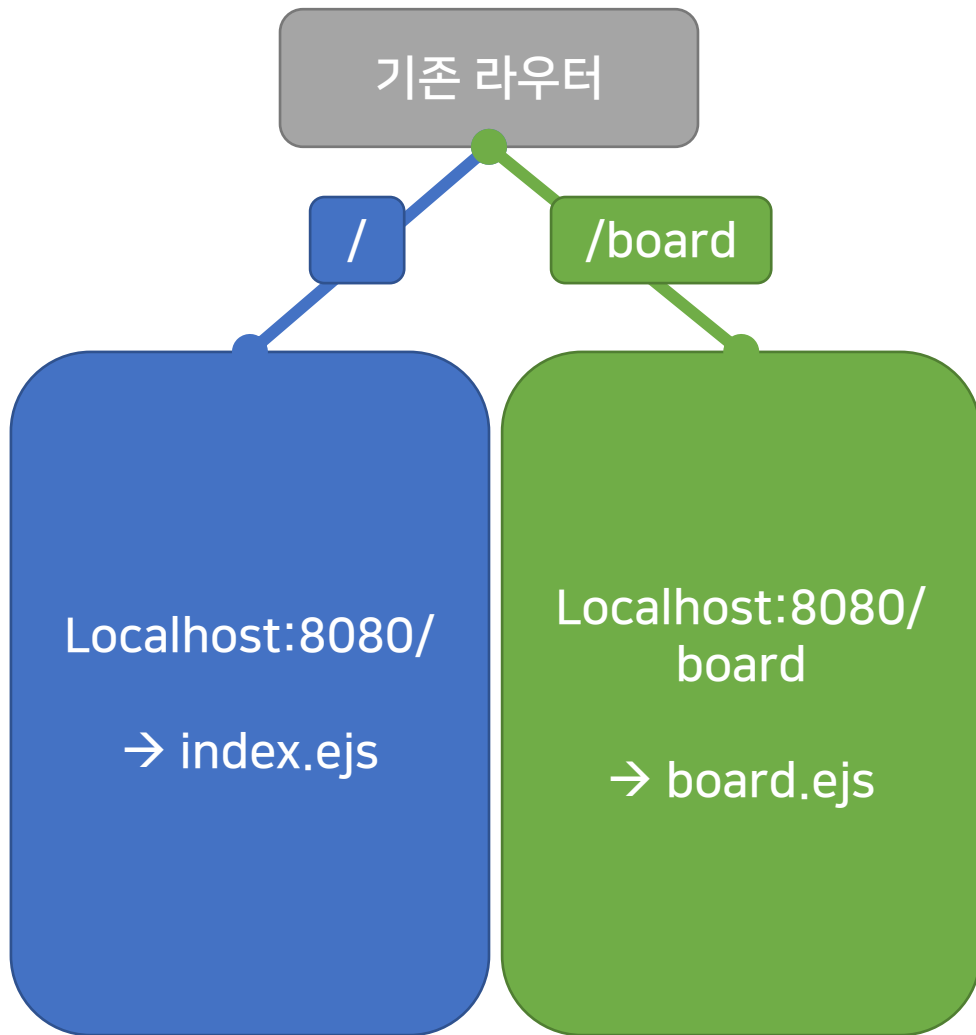
src/components/FancyBorder.js

```
import FancyBorder from "../components/FancyBorder";  
  
function App() {  
  return (  
    <div className="App">  
      <FancyBorder color="blue">  
        <h1>Helle, props.children</h1>  
        <p>It's so convinient tools for us</p>  
      </FancyBorder>  
    </div>  
  );  
}
```

src/App.js



React Router





React Router 모듈 설치 및 적용

- Npm install react-router-dom
- 그리고 index.js 컴포넌트로 가서 App 컴포넌트를 `<BrowserRouter>` 라는 react-router-dom 이라는 컴포넌트로 감싸 주세요!
- `<BrowserRouter>` 로 감싸 주어야만 `<App>` 컴포넌트에서 발생하는 주소 값의 변경을 감지 할 수 있습니다
- 그외의 라우터로는 `<HashRouter>` 가 유명하며 주소의 해시 주소 `localhost:3000/#hash` 를 감지할 수 있는 라우터 입니다!



```
import { BrowserRouter } from 'react-router-dom';

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

reportWebVitals();
```

Src/index.js



```
import { Link, Outlet, Route, Routes } from "react-router-dom";
import Profile from "../components/Profile";
import Board from "../components/Board";
```

```
function App() {
  return (
    <div className="App">
      <nav>
        <ul>
          <li>
            <Link to="/">홈 페이지 이동</Link>
          </li>
          <li>
            <Link to="/profile">프로필 페이지 이동</Link>
          </li>
          <li>
            <Link to="/board">게시판 페이지 이동</Link>
          </li>
        </ul>
      </nav>
      <Routes>
        <Route path="/profile" element={<Profile />} />
        <Route path="/board" element={<Board />} />
      </Routes>
    </div>
  );
}
```

```
export default App;
```

전체 코드

Src/App.js



Localhost:3000/



[홈 페이지 이동](#)
[프로필 페이지 이동](#)
[게시판 페이지 이동](#)

Localhost:3000/profile



[홈 페이지 이동](#)
[프로필 페이지 이동](#)
[게시판 페이지 이동](#)

프로필 페이지 입니다

Localhost:3000/board



[홈 페이지 이동](#)
[프로필 페이지 이동](#)
[게시판 페이지 이동](#)

게시판 페이지 입니다

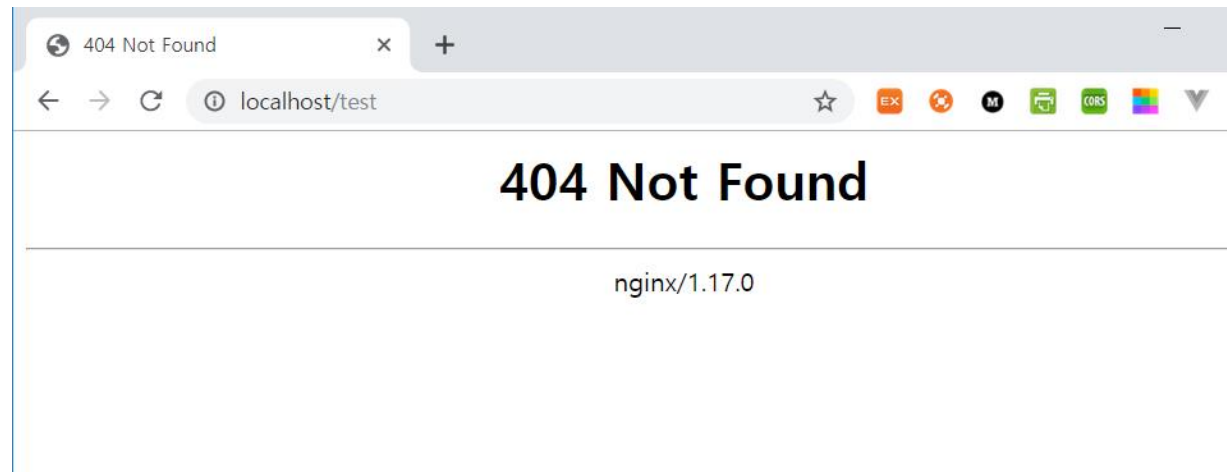


주소 예외 처리



Page Not Found

- 실제 서비스의 경우 사용자가 예상하지 못한 주소 값을 입력하는 경우가 발생 합니다.
- 이럴 때, 브라우저에서 제공하는 **404 Not Found** 페이지를 띄우면 일단 서버 응답을 기다리는 시간도 오래 걸릴 뿐더러, 결과 페이지가 서비스의 신뢰를 깨는 역할을 하게 됩니다!





Page Not Found

- 따라서 잘못 입력 된 주소에 대한 예외 처리가 필요합니다!
- React-router-dom 모듈은 해당 부분에 있어서 * 라는 편리한 방법을 제공합니다!
- * 는 모든 주소 입력을 의미하며 아래와 같이 사용합니다!

```
<Route path="*" element={<NotFound />} />
```



```
function App() {  
  return (  
    <div className="App">  
      <Routes>  
        <Route path="/" element={<Header />} />  
        <Route path="profile" element={<Profile />} />  
        <Route path="board" element={<Board />} />  
        <Route path="*" element={<NotFound />} />  
      </Routes>  
    </div>  
  );  
}  
  
export default App;
```

코드 처리 방향

Src/App.js



주소

Parameter 활용



Route 에 Parameter 선언

- App.js 에 선언 된 라우터 선언부에 Parameter 를 선언해 봅시다!
- 기존과 같은 방법으로 주소/:parameter 로 선언하면 됩니다!

```
<Routes>
  <Route path="/" element={<Header />} />
  <Route path="profile" element={<Profile />} />
  <Route path="board" element={<Board />} />
  <Route path="board/:boardID" element={<BoardDetail />} />
  <Route path="*" element={<NotFound />} />
</Routes>
```

Src/App.js



```
import { useParams } from "react-router-dom"
import Header from "../Header";
```

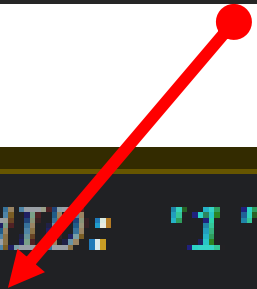
```
export default function BoardDetail() {
  const params = useParams();
  console.log(params);
  return (
    <>
      <Header />
      <h2>{params.boardID} 번 게시글 내용입니다!</h2>
    </>
  )
}
```

Src/component/BoardDetail.js



전달한 이름 그대로 객체의 Key 값이 배정

```
<Route path="board/:boardID" element={<BoardDetail />} />
```

A red arrow originates from the `boardID` prop in the code above and points to the `boardID` key in the object below.

```
▼ {boardID: '1'} ⓘ  
  boardID: "1"  
  ► [[Prototype]]: Object
```



Redux



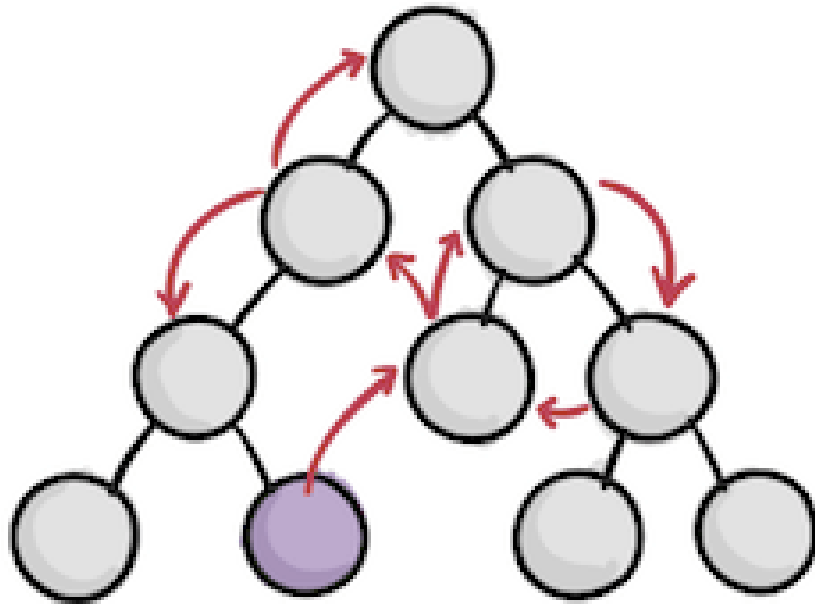
코딩애플

구독자 10.2만명

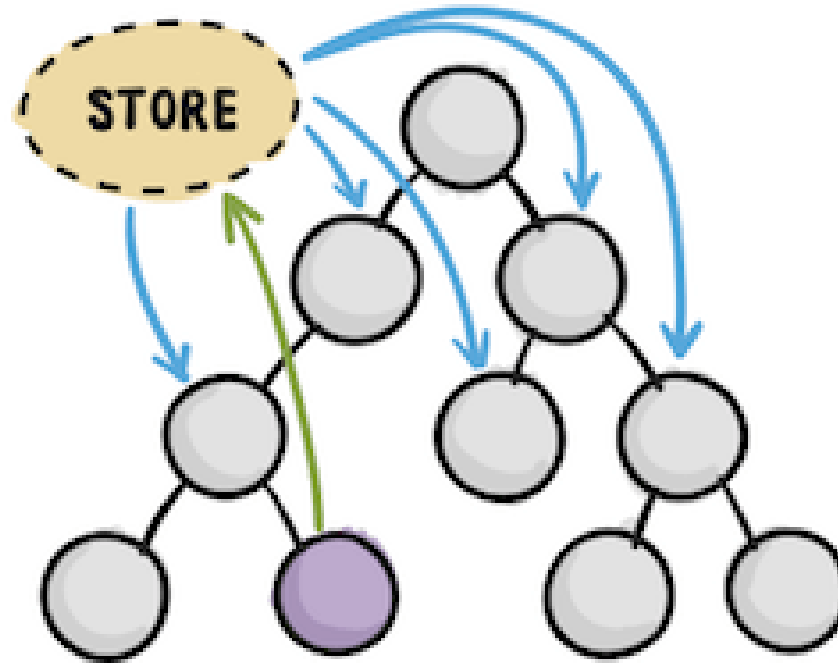
<https://www.youtube.com/watch?v=QZcYz2NrDIs&t=212s>



WITHOUT REDUX

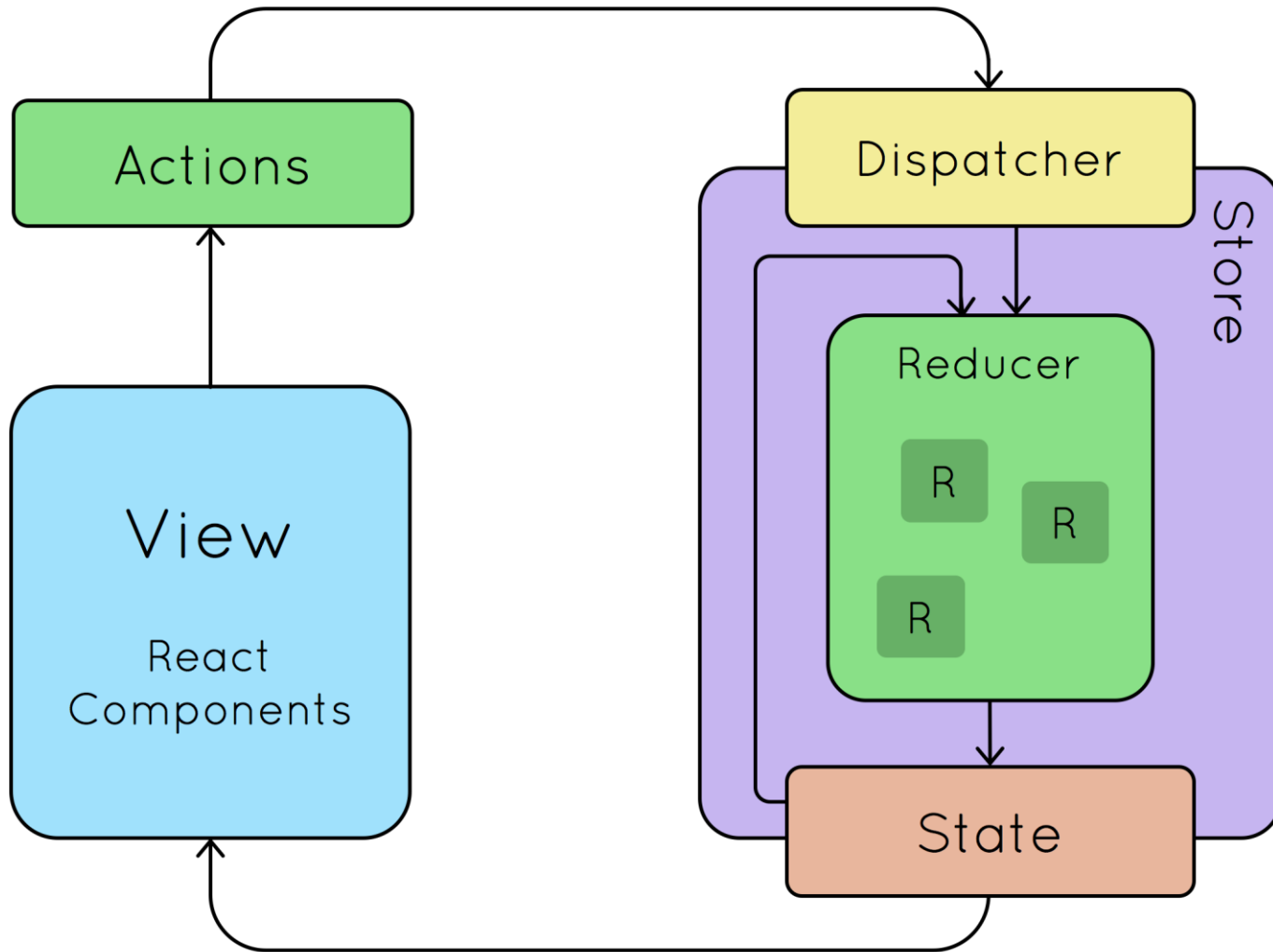


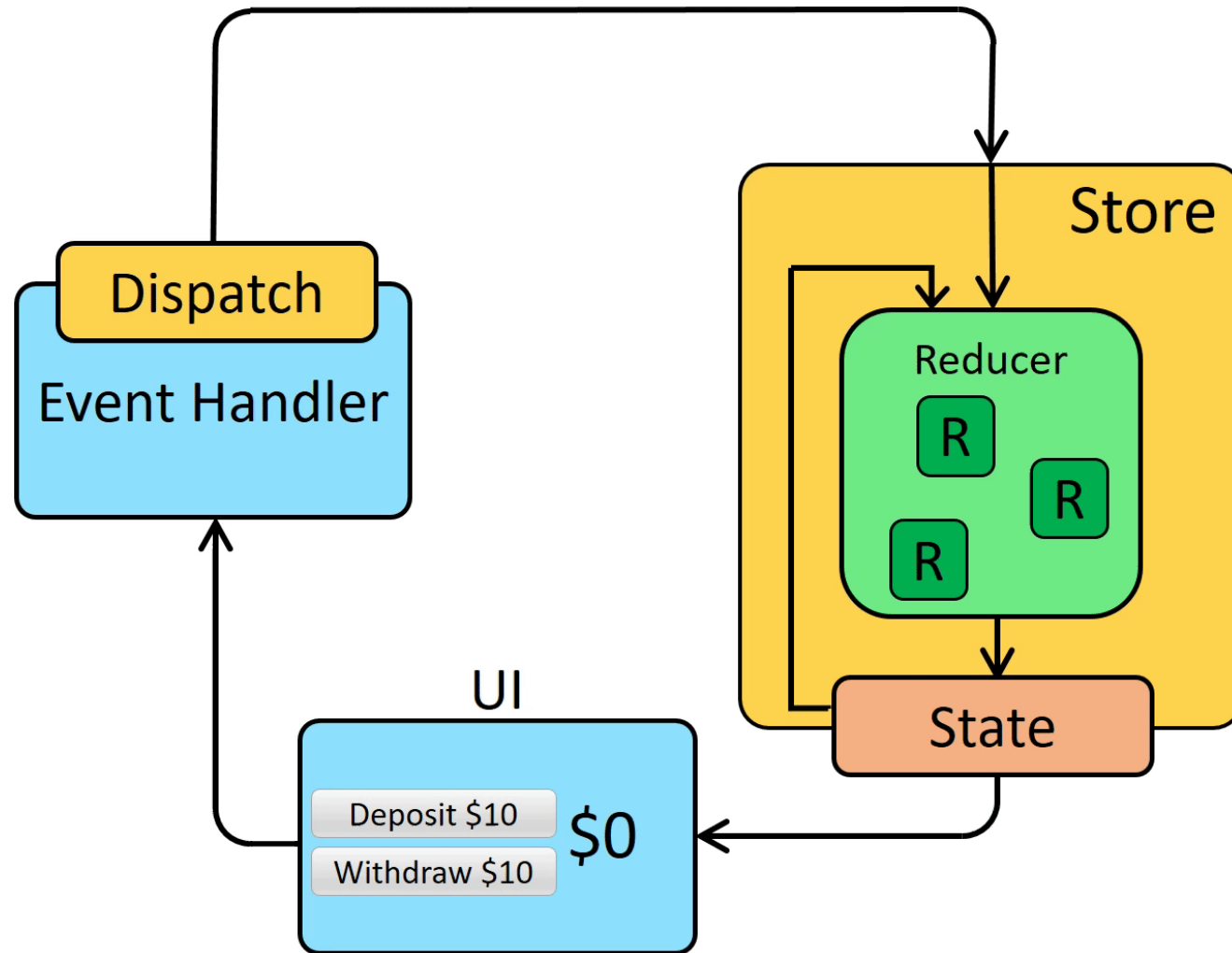
WITH REDUX



 COMPONENT INITIATING CHANGE

Redux 동작 순서







코딩 애플 코드로 하나하나 확인하기



Redux 기초 세팅!

- Redux 를 위한 새로운 App 을 만듭시다!
- `Npx create-react-app redux-app`
- 앱 생성 후, redux 관련 모듈을 설치 합시다!
- `Npm install redux`
- `Npm install react-redux`

```
import { Provider } from 'react-redux';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <>
    <Provider>
      <App />
    </Provider>
  </>
);
```

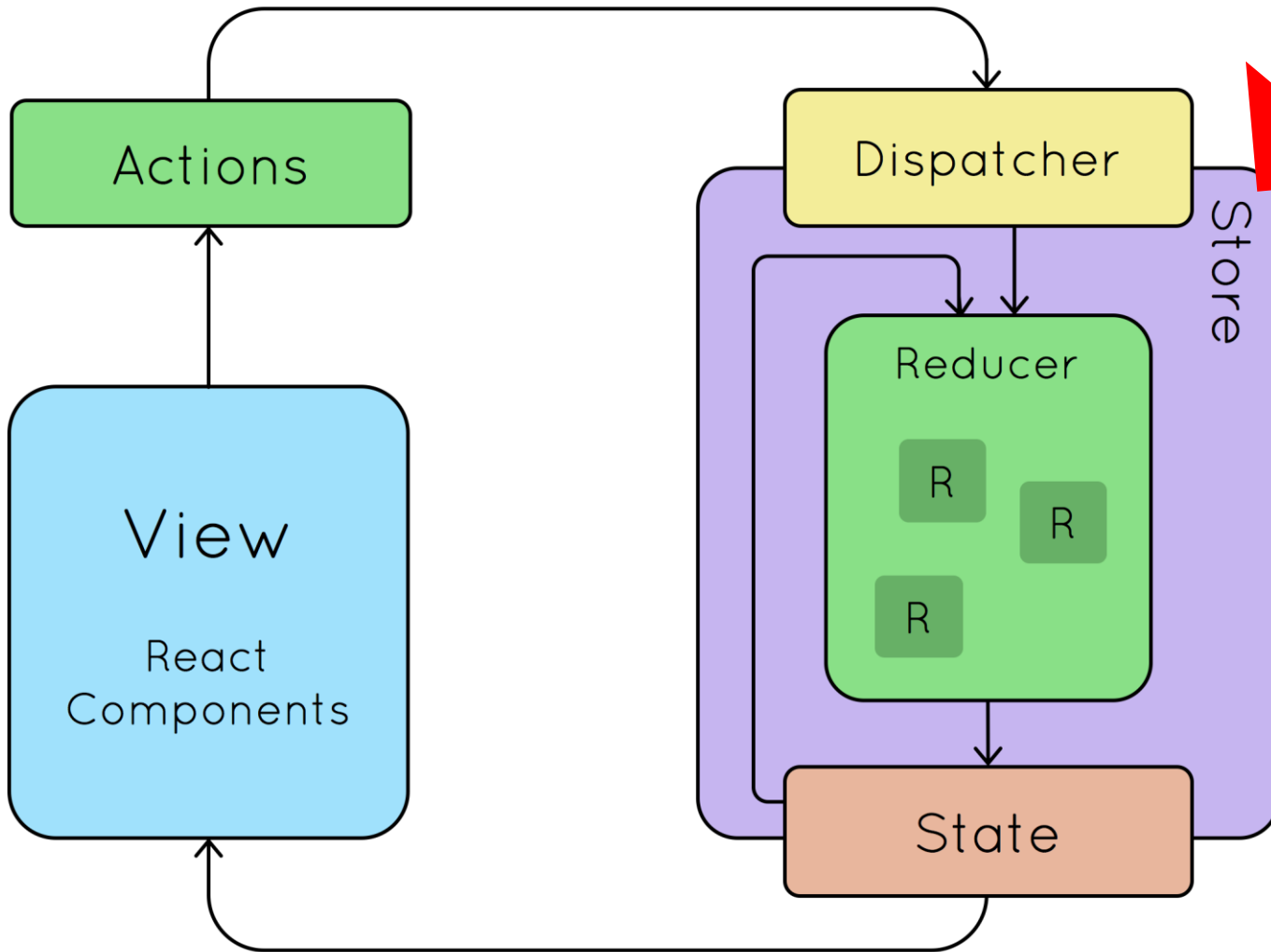
Src/index.js





Store

Store 만들기!



- 상태를 저장할 Store 부터 만들어 봅시다!



```
import { createStore } from 'redux';
```

```
let store = createStore(reducer);
```

```
const root = ReactDOM.createRoot(document.getElementById('root'));
```

```
root.render(  
  <>  
    <Provider store={store}>  
      <App />  
    </Provider>  
  </>  
);
```

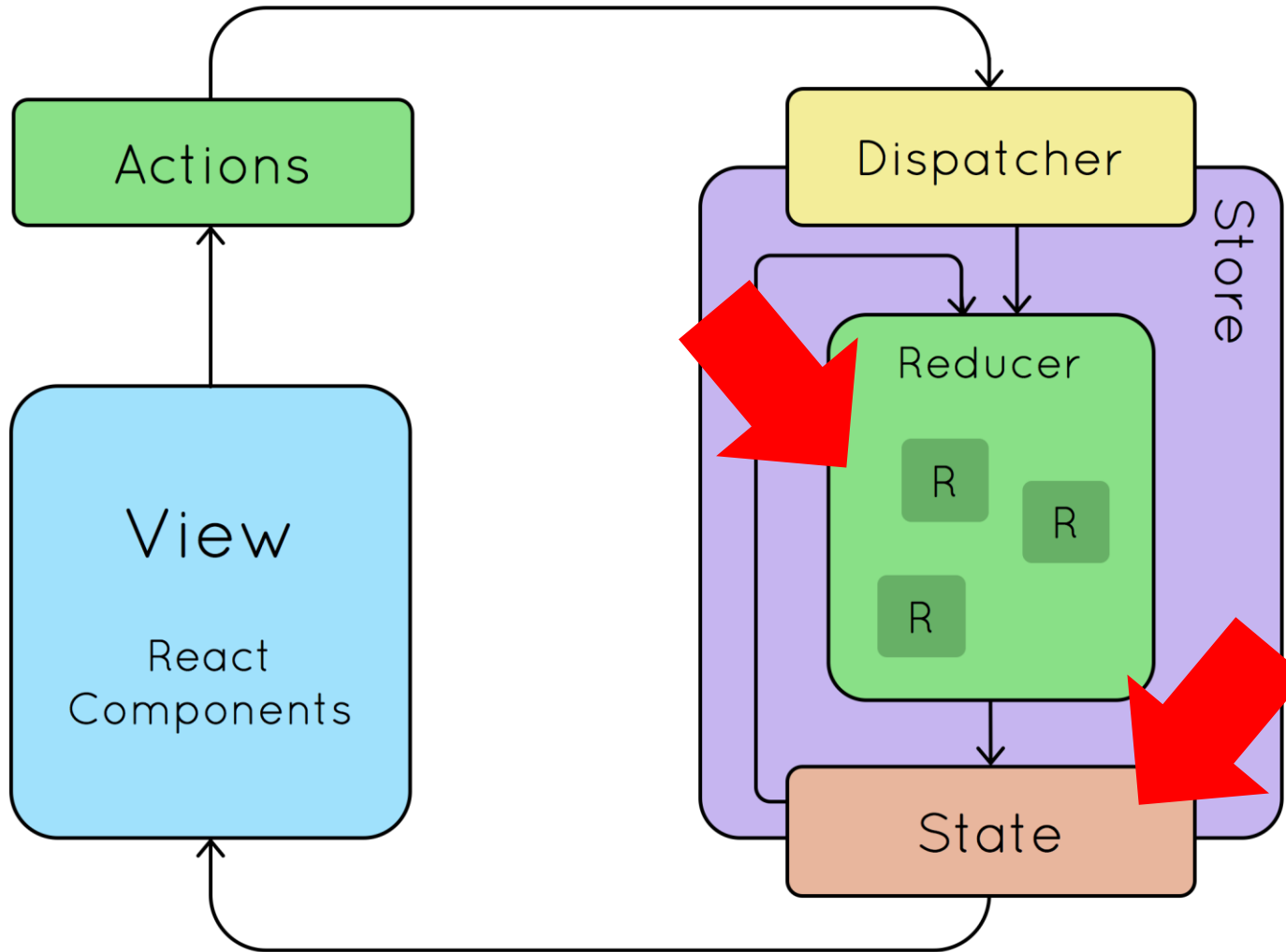
Src/index.js



State & Reducer



State 값 설정 및 Reducer 만들기!



- 상태 역할을 해줄 State 값 설정을 하고
- State 값을 변경해 줄 Reducer 를 만들어 봅시다!



```
let weight = 100;

function reducer(state = weight) {
  return state;
}

let store = createStore(reducer);

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

reportWebVitals();
```

Src/index.js

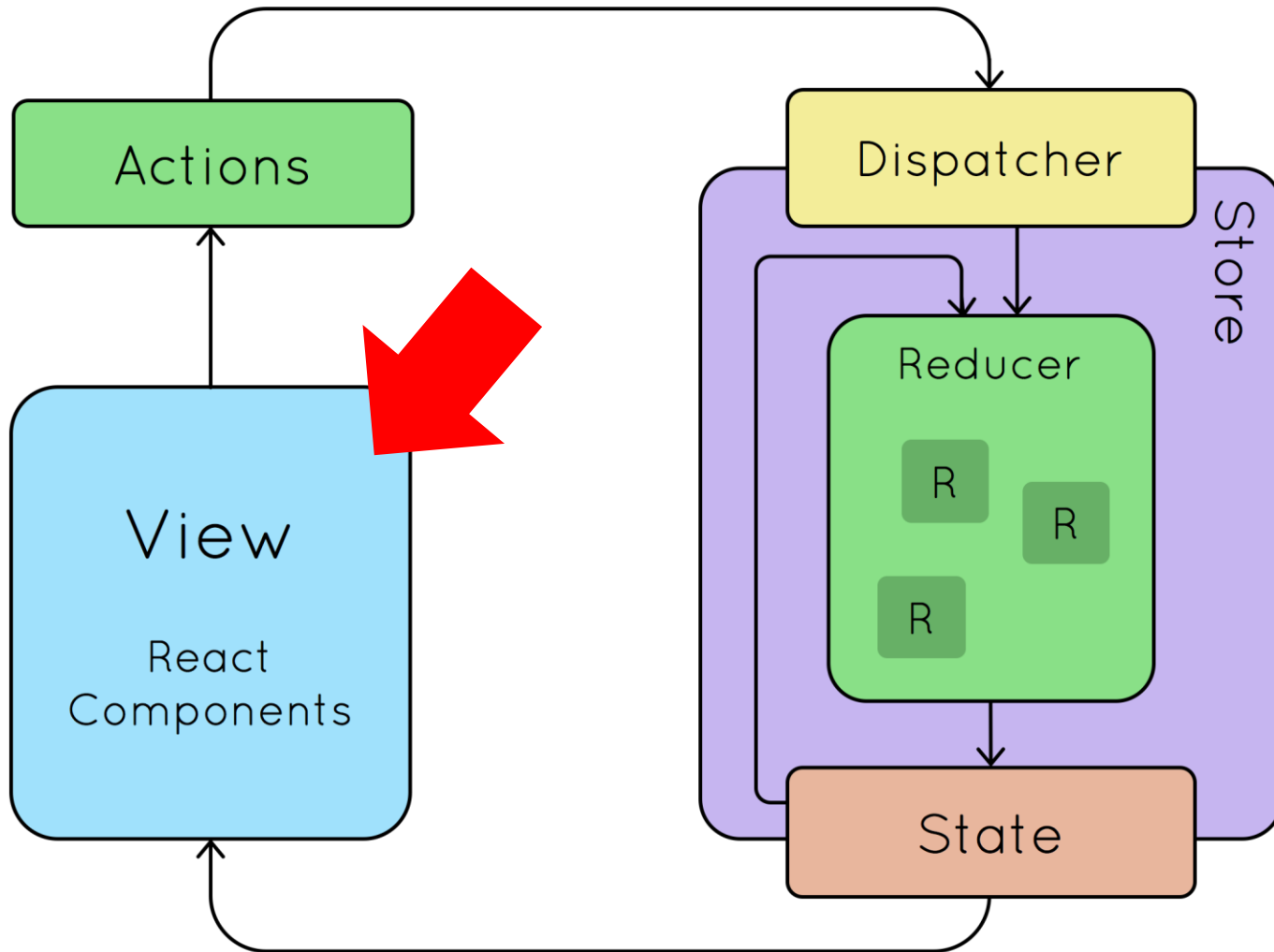
- 아직은 Action 에 따라 상태 값을 변경하는 기능은 없고, index.js 에 저장된 상태 값을 전역에 있는 컴포넌트에 전달하는 기능만을 수행 합니다!



Store 에 저장 된
값 받아오기!



Store 에 저장 된 값 받아오기!



- App.js 내부의 컴포넌트에서 Store 에 저장 되어있던 State 값을 받아 옵시다!



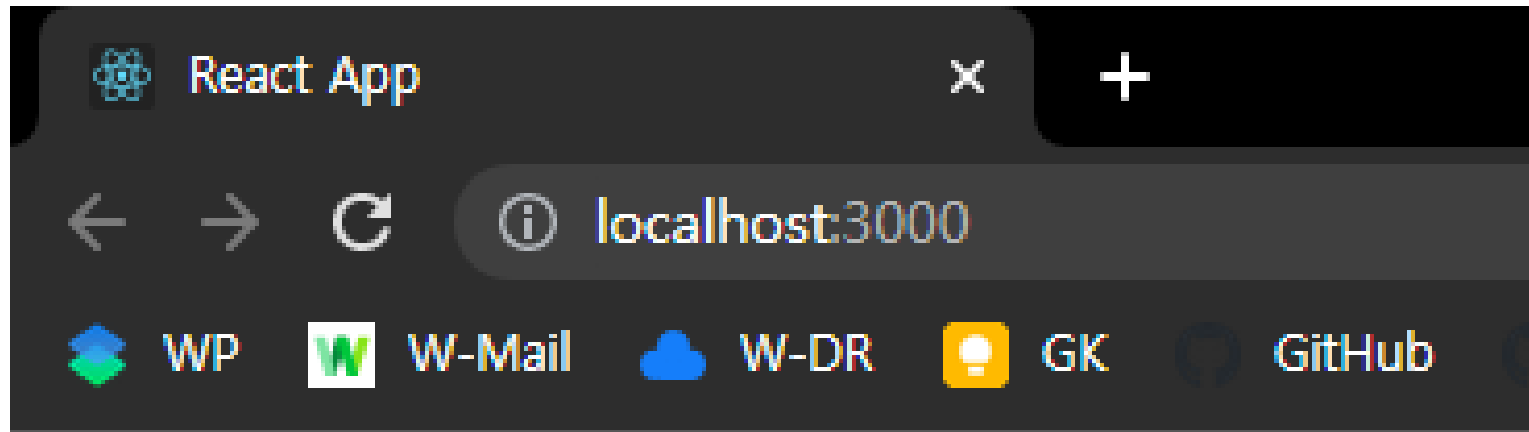
```
import React from 'react'
import { useSelector } from 'react-redux'

export default function Test() {
  const weight = useSelector((state) => state);

  return (
    <>
      <h1>당신의 몸무게는 {weight}</h1>
    </>
  )
}
```

Src/component/Test.js

- weight 라는 변수에 Store 에 저장 되어있던 상태 값을 받고, 활용하면 됩니다!

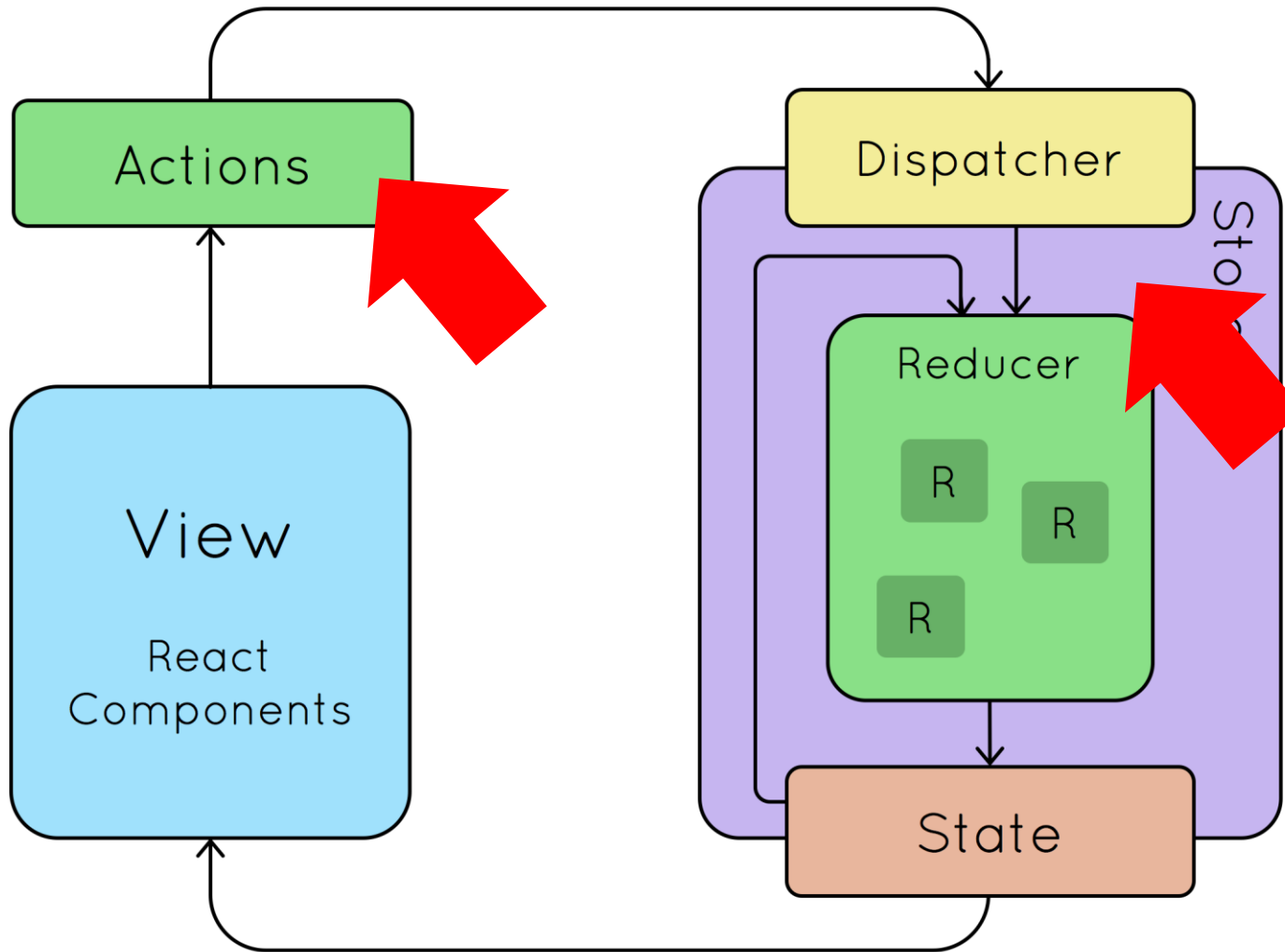


당신의 몸무게는 100



Action & Reducer

Action 설정과 Dispatch 로 Action 보내기



- Store 의 State 값 변경을 위해서는 Action 을 설정
- Action 을 Dispatch 를 사용해서 Reducer 에 전달
- Reducer 가 State 를 변경



```
function reducer(state = weight, action) {  
  if (action.type === "증가") {  
    state++;  
    return state;  
  } else if (action.type === "감소") {  
    state--;  
    return state;  
  } else {  
    return state;  
  }  
}
```

Src/index.js



Dispatch



```
import React from 'react'
import { useSelector, useDispatch } from 'react-redux'

export default function Test() {
  const weight = useSelector((state) => state);
  const dispatch = useDispatch();

  return (
    <>
      <h1>당신의 몸무게는 {weight}</h1>
      <button onClick={() => { dispatch({ type: "증가" }) }}>살 찌기</button>
      <button onClick={() => { dispatch({ type: "감소" }) }}>살 빼기</button>
    </>
  )
}
```

Src/component/Test.js



당신의 몸무게는 100

살 찌기

살 빼기

당신의 몸무게는 105

살 찌기

살 빼기

당신의 몸무게는 92

살 찌기

살 빼기



지금 시작합니다



실전이야!



그래서 우리가
만들 것은!?



What's the Plan for Today?

Add Todo

Like and Subscribe!



Wash the Dishes



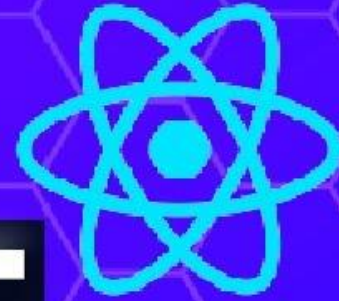
Walk the Gorilla



Skydive over Everest



REACT TODO LIST



Todo 리스트 만들기!



- 정말 간단하게 만들 겁니다!
- 할 일 추가, 할 일 완료를 누르면 완료 목록으로 옮기는 기능 정도만 추가해 볼게요!
- 리액트 앱 부터 만들고 시작하시죠!



기초 세팅!

- 먼저 새롭게 만들 app 을 만들어 봅시다!
 - `Npx create-react-app todo-app`
- 필요 모듈을 한큐에 설치!
 - `npm i redux react-redux @reduxjs/toolkit`



Store

폴더 생성



Store 폴더 생성

- 기능 이전에 저장할 근간부터 만들어야 겠죠?
- Src 폴더 내부에 store 폴더를 만들어 주세요!
- Store 전체를 총괄하는 모듈은 `index.js` 가 담당할 예정입니다!



>	node_modules	
>	public	
▼	src	●
>	components	●
▼	store	●
	JS index.js	U
#	App.css	
JS	App.js	M
JS	App.test.js	
#	index.css	
JS	index.js	M
🖼️	logo.svg	
JS	reportWebVitals.js	
JS	setupTests.js	
📄	.gitignore	
{ }	package-lock.json	M
{ }	package.json	M
📖	README.md	

Store 모듈 분할

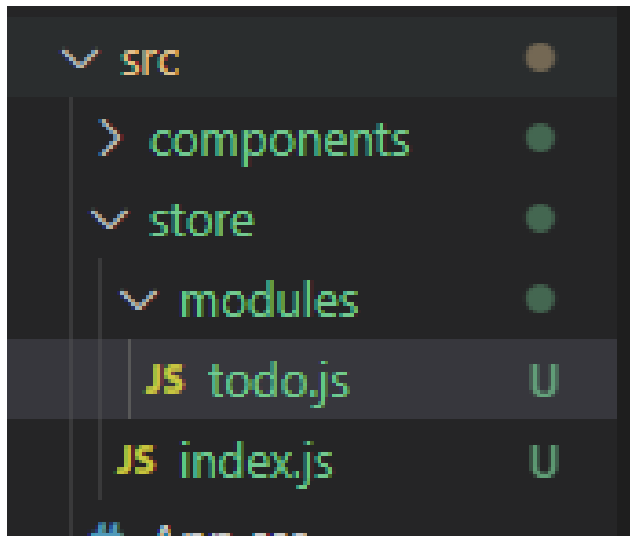


- 모든 컴포넌트에 대한 글로벌 상태 값을 하나의 파일에서 관리한다면?
- 해당 파일이 하는 일이 너무 많겠죠?
- 당연히 코드 확장성 및 관리에도 어려움이 생깁니다! → 각 기능별 Store 모듈을 분할 합니다!



Store 모듈 분할

- 먼저 store 내부에 modules 폴더를 만들어 봅시다!
- 우리는 ToDo List 를 만들 것이므로 해당 리스트를 관리하는 모듈인 **todo.js** 모듈을 modules 폴더 내부에 만들어 줍시다!





초기 State 값 선언하기



최초의 State 값 설정

- 컴포넌트가 최초 렌더링 될 때 보여줘야할 최초의 State 값을 설정해 봅시다!
- 물론 DB 에서 데이터를 받아서 설정해 주는 방법이 맞지만, 이전 백엔드와 마찬가지로 편의를 위해서 변수로 설정해 봅시다!



최초의 State 값 설정

- Todo List 이므로 객체가 담긴 배열 형태로 선언할 예정입니다!
- List 객체에는 아래의 값이 구성 될 예정입니다!
 - **id**: 고유 id 값
 - **text**: 할 일 내용
 - **done**: 완료 여부



```
// 초기 상태 설정
const initState = {
  list: [
    {
      id: 0,
      text: '리액트 공부하기',
      done: false,
    },
    {
      id: 1,
      text: '척추의 요정이 말합니다! 척추 펴기!',
      done: false,
    },
    {
      id: 2,
      text: '취업 하기',
      done: false,
    },
  ],
};
```

Src/store/modules/todo.js



Reducer 로
값 리턴 시키기!



Reducer 를 통해 State 전달!

- 설정한 State 값을 외부에서 접근 하기 위해서는 Reducer 를 통해 값을 return 시켜줘야 합니다!
- 설정한 State 값을 바로 return 시켜주는 간단한 Reducer 를 작성해 봅시다!

```
export default function todo(state = initState, action) {  
  return state;  
}
```

Src/store/modules/todo.js



- 원래는 전달 된 2번째 매개 변수인 action 의 type 에 따라 다른 동작을 수행하는 것이 진짜 Reducer 입니다.
- 지금은 초기 State 값을 외부로 전달하는 목적만 달성하면 되므로 state 매개 변수에 initState 값을 넣어서 바로 return 시켜 주면 됩니다!



Store

통합 관리



Store 통합 관리!

- Store 는 모듈 별로 관리하고, 모듈 들은 Store 폴더의 **index.js** 에 의해서 통합 관리 됩니다!
- Store 폴더의 **Index.js** 파일에 가서 모듈 들을 통합 관리 해봅시다!
- 먼저 초기 값을 선언한 todo.js 를 import 해서 todo.js 의 reducer 를 불러오고(export default 로 설정 하였음) redux 의 **combineReducer** 를 이용하여 todo.js 의 reducer 를 하나로 합쳐서 다시 내보내 줍시다!



```
// 통합 관리 파일
import { combineReducers } from 'redux';
import todo from './modules/todo';

export default combineReducers({
  todo,
});
```

Src/store/index.js

- 각각의 Reducer 들을 합쳐주는 **combineReducer** 를 이용해서 각각의 store 모듈에서 export 된 reducer 를 합쳐 줍시다!
- 그리고 다시 합쳐진 reducer 를 export default 로 보내내 줍시다!



Redux 기초 세팅 및 Store 연결



Redux 기초 세팅!

- 라우팅 처리 하던 것처럼! Redux 적용을 위해서는 `<Provider>` 컴포넌트를 임포트하고 해당 컴포넌트로 `<App>` 컴포넌트를 감싸줘야만 합니다!
- Src 폴더의 최상위 `Index.js` 에 가서 코드를 처리!(Store 의 index.js X)
- `combineReducer` 를 통해 하나로 합쳐서 내보낸 Reducer 는 `rootReducer` 라는 값으로 받아 줍시다!



```
import { Provider } from 'react-redux';
import { configureStore } from '@reduxjs/toolkit';
import rootReducer from './store';

const store = configureStore({ reducer: rootReducer });

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

Src/index.js



TodoList

컴포넌트 작성



Todo List 의 기본이 될 컴포넌트 만들기!

- 할 일 목록을 보여주고, 추가하는 기능을 가지는 → `<TodoList>` 컴포넌트
- 완료된 목록을 보여주는 → `<DoneList>` 컴포넌트
- 위의 두 컴포넌트를 “포함” 하여 전체 앱을 그려주는 `<ListContainer>` 컴포넌트를 제작해 봅시다!



TodoList

컴포넌트



<TodoList> 컴포넌트

- 할 일을 추가하는 Input 요소와, 추가 버튼 요소를 만들어 줍시다!
- 할 일 목록을 redux 를 통해 Store 에서 받아온 다음, 해당 목록을 `` 태그의 `` 요소로 그려 줍시다!



```
import { useRef } from 'react';
import { useSelector } from 'react-redux';

export default function TodoList() {
  const list = useSelector((state) => state.todo.list);
  const inputRef = useRef();

  return (
    <section>
      <h1>할일 목록</h1>
      <div>
        <input type="text" ref={inputRef} />
        <button>추가</button>
      </div>
      <ul>
        {list.map((el) => {
          return <li key={el.id}>{el.text} <button>완료</button> </li>;
        })}
      </ul>
    </section>
  );
}
```

Src/component/TodoList.jsx



DoneList

컴포넌트



<DoneList> 컴포넌트

- List 는 일단 useSelector 를 이용해서 state 값을 받아 옵시다!
- 완료된 List 를 받으면, 해당 List 를 ui 요소로 그려주면 도비니다!
- <TodoList> 에서 인풋 입력을 뺀 상태로 비슷하게 구현하면 됩니다!



```
import { useSelector } from "react-redux";

export default function DoneList() {
  const list = useSelector((state) => state.todo.list);
  return (
    <section>
      <h1>완료된 목록</h1>
      <ul>
        {list.map((el) => {
          return (
            <li key={el.id}>
              {el.text}
            </li>
          );
        })}
      </ul>
    </section>
  );
}
```

Src/component/DoneList.jsx



ListContainer

컴포넌트



<ListContainer> 컴포넌트

- <ListContainer> 컴포넌트는 <TodoList> 와 <DoneList> 를 순서대로 포함만 하면 되므로, 각각 컴포넌트를 Import 한 다음 자식 요소로 만들어 줍니다!



```
import TodoList from './TodoList';
import DoneList from './DoneList';

export default function ListContainer() {
  return (
    <>
      <TodoList />
      <DoneList />
    </>
  );
}
```

Src/component/ListContainer.jsx



할일 목록

dasdadas

추가

- 리액트 공부하기
- 척추의 요정이 말합니다! 척추 펴기!
- 취업 하기

완료된 목록

- 리액트 공부하기 완료
- 척추의 요정이 말합니다! 척추 펴기! 완료
- 취업 하기 완료



Redux 를 위한 편의 도구



편의 도구가 왜 필요하죠?

- Redux 에 저장 된 Store 의 값은 src 폴더의 `index.js` 파일에서 `getState()` 메소드를 이용하여 확인을 합니다!
- 지금은 간단한 Todo List 이기 때문에 하나의 값 만을 처리하고 있지만 프로젝트가 커지게 되면 다양한 전역 상태 값을 redux 로 관리 해야 합니다.
- 그럴때 마다 redux 의 모든 값을 하나하나 `console.log` 로 찍어가면서 확인하기는 매우 귀찮기 때문에 사용합니다!



```
import { Provider } from 'react-redux';
import { configureStore } from '@reduxjs/toolkit';
import rootReducer from './store';

const store = configureStore({ reducer: rootReducer });
console.log(store.getState());

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

Src/index.js



▼ Object ⓘ

▼ todo:

▼ list: Array(3)

▶ 0: {id: 0, text: '리액트 공부하기', done: false}

▶ 1: {id: 1, text: '척추의 요정이 말합니다! 척추 펴기!', done: false}

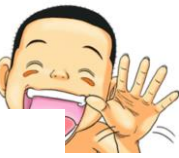
▶ 2: {id: 2, text: '취업 하기', done: false}

length: 3

▶ [[Prototype]]: Array(0)

▶ [[Prototype]]: Object

▶ [[Prototype]]: Object



홈 > 확장 프로그램 > Redux DevTools



Redux DevTools

📍 추천

★★★★★ 571 ⓘ | 개발자 도구 | 사용자 1,000,000+명

Chrome에 추가

<https://chrome.google.com/webstore/detail/redux-devtools/lmhkpmbekcpmknklieibfkpmmfibljd?hl=ko>

사용법!



- <https://github.com/reduxjs/redux-devtools/tree/main/extension#installation>

1.1 Basic store

For a basic Redux store simply add:

```
const store = createStore(  
  reducer, /* preloadedState, */  
+ window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__()  
);
```

사용법!



- redux store 를 만들 때, 약속 된 코드를 삽입해 주면 됩니다!

```
const reduxDevTool =  
  window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__();  
  
const store = configureStore({ reducer: rootReducer }, reduxDevTool);
```

Src/index.js

Actions Settings

filter...

@@INIT 8:14:01.26

State Action State Diff Trace Test

Tree Chart Raw

- ▼ todo (pin)
 - ▼ list (pin)
 - ▶ 0 (pin): { id: 0, text: "리액트 공부하기", done: false }
 - ▶ 1 (pin): { id: 1, text: "책추의 요정이 말함...", done: false }
 - ▶ 2 (pin): { id: 2, text: "취업 하기", done: false }

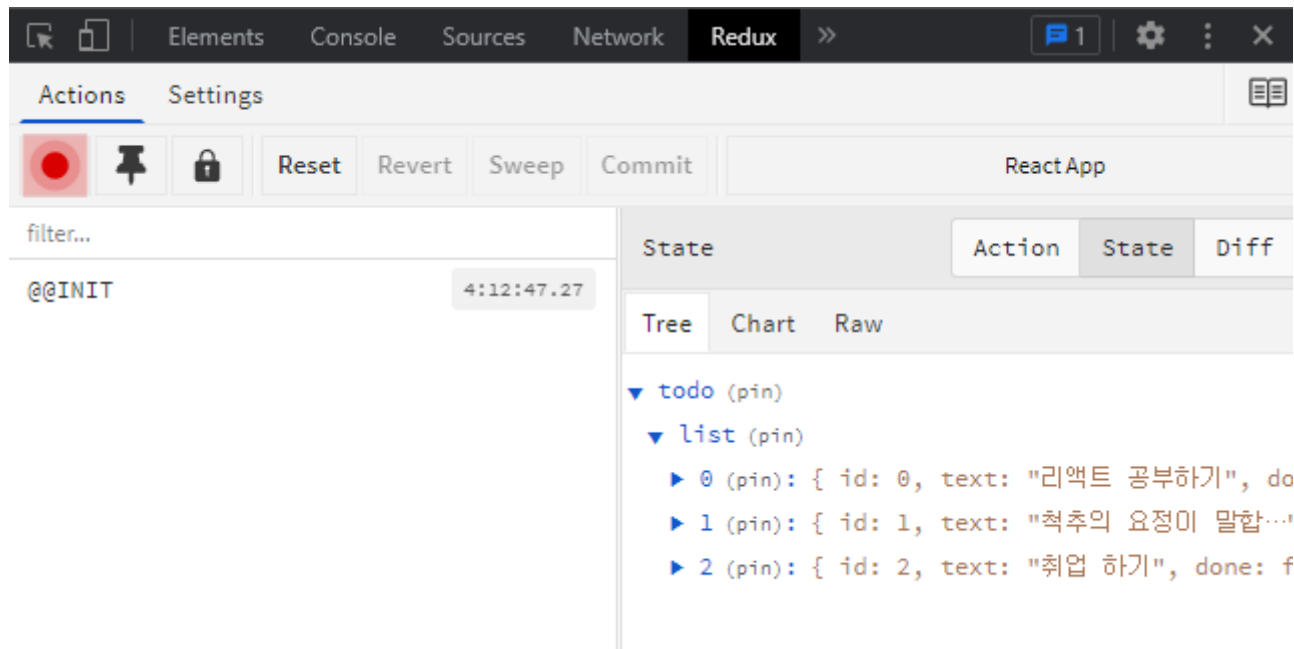
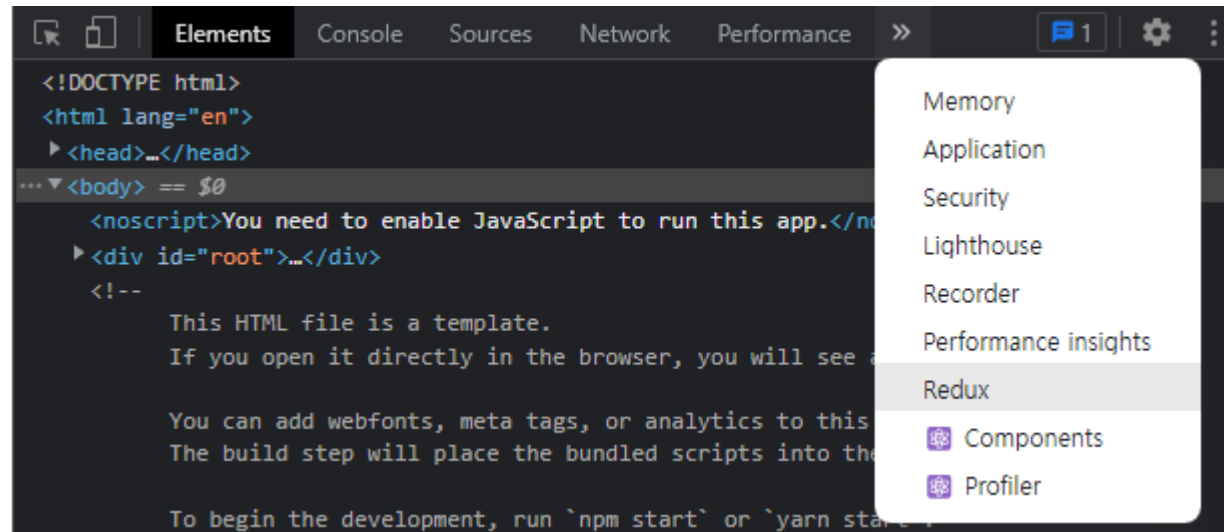
@@INIT (0)

Inspector Log monitor Chart RTK Query

React App

Live 1x 2x







Action 타임

설정



Action 타입 정의하기

- Action 타입은 "문자열"로 보통 정의합니다!
- Todo 리스트에 필요한 생성, 완료 액션을 정의합니다!

```
// 액션 타입 정의하기  
const CREATE = "todo/CREATE";  
const DONE = "todo/DONE";
```

Src/store/modules/todo.js



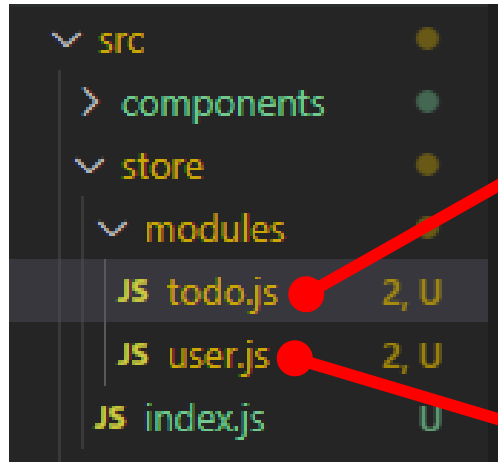
Action 타입 정의하기

- 앞에 **todo/** 는 왜 붙이는 건가요?
- 이 것은 잘못 된 사용을 막기 위한 하나의 방법입니다!
- 모듈이 달라도 Action 타입으로 CREATE, DELETE, DONE 같은 변수명은 상당히 많이 사용이 됩니다.
- 그럴 때 잘못 된 모듈 Import 로 인해, 다른 Reducer 의 기능이 호출되면 문제가 발생합니다



Action 타입 정의하기

- 이럴 때 Action 타입 앞에 지금 이 액션의 타입이 어떤 모듈의 타입인지를 알려주는 문자열을 추가하여 위와 같은 문제가 발생하는 것을 막아 줍니다!



```
// 액션 타입 정의하기  
const CREATE = "todo/CREATE";  
const DONE = "todo/DONE";
```

```
// 액션 타입 정의하기  
const CREATE = "user/CREATE";  
const DONE = "user/DONE";
```



Action 생성 함수

작성



Action 생성 함수 작성

- 외부 컴포넌트에서 Action 을 만들어주는 함수부터 작성을 해봅시다!
- Action 생성 함수는 type 정보와 전달해야 할 정보를 payload 객체에 담아서 **Dispatch** 를 통해 전달 합니다!
- 결과적으로 Reducer 가 Action 함수에 들어있는 type 을 확인해서 어떤 행동을 할지 정하고, payload 에 있는 데이터를 받아서 처리 합니다!



Action 생성 함수 작성 - Create

- 새로운 할 일 목록을 만드는 create 함수부터 작성해 봅시다!
- 먼저 Action type 설정 부터 CREATE 로 해줍니다!
- 그리고 전달 해야할 정보는 payload 라는 매개 변수에 담아서 전달 합니다!



```
// 액션 생성 함수 작성
export function create(payload) {
  return {
    type: CREATE,
    payload,
  };
}
```

Src/store/modules/todo.js



Action 생성 함수 작성 - Done

- 할 일을 완료하는 역할을 하는 Done 함수도 작성해 봅시다!
- 먼저 Action type 설정 부터 DONE 으로 해줍니다!
- 이번에는 새로운 정보를 전달 할 필요가 없이 어떤 목록이 완료 되었는지만 알면 되기 때문에 id 값만 전달 하면 됩니다!

```
export function done(id) {  
  return {  
    type: DONE,  
    id,  
  };  
}
```

Src/store/modules/todo.js





```
// 액션 타입 정의하기
const CREATE = 'todo/CREATE';
const DONE = 'todo/DONE';

// 액션 생성 함수 작성
export function create(payload) {
  return {
    type: CREATE,
    payload,
  };
}

export function done(id) {
  return {
    type: DONE,
    id,
  };
}
```

외부에서 직접 요청하지 않고
Todo.js 의 함수를 import 해서
사용하는 이유는

이렇게 type 값 등을 외부에서는
알 수가 없기 때문에
약속 된 함수만 사용하여
접근하는 것이 편하기 때문입니다!

Src/store/modules/todo.js

전체 코드



Reducer

구조 구현



Action Type 에 따라 작동하는 Reducer

- 이제는 Action Type 에 따라 작동하는 Reducer 를 구현해 봅시다!
- 먼저, switch 문을 이용해서 action type 에 따라서 각각의 역할을 한 뒤 값을 return 하는 구조로 만들어 주시면 됩니다!



// 리듀서 설정(실제 작업은 이친구가 합니다!)

```
export default function todo(state = initState, action) {  
  switch (action.type) {  
    case CREATE:  
      return console.log('CREATE 호출');  
    case DONE:  
      return console.log('DONE 호출');  
    default:  
      return state;  
  }  
}
```

Src/store/modules/todo.js



Dispatch 로 Action 함수 전달



Dispatch 로 Action 함수 전달

- 그럼 이번에는 컴포넌트에서 **Dispatch** 로 정의한 Action 함수를 Reducer 에 전달하여 정상적으로 호출이 되는지 확인해 봅시다!
- Dispatch 활용을 위해 **useDispatch** 를 dispatch 변수에 넣어주기!
- **Src/store/modules/todo.js** 에서 create, done 함수 불러오기!
- **Dispatch** 의 인자로 create, done 함수를 전달하여 호출 상태 확인!



```
import { useRef } from 'react';  
import { useDispatch, useSelector } from 'react-redux';  
import { create, done } from '../store/modules/todo';
```

```
export default function TodoList() {  
  const list = useSelector((state) => state.todo.list);
```

```
  const inputRef = useRef();  
  const dispatch = useDispatch();
```

```
  return (  
    <section>  
      <h1>할일 목록</h1>  
      <div>  
        <input type="text" ref={inputRef} />  
        <button  
          onClick={() => {  
            dispatch(create(''));  
          }}  
        >  
          추가  
        </button>  
      </div>  
      <ul>  
        {list.map((el) => {  
          return <li key={el.id}>{el.text}</li>;  
        })}  
      </ul>  
    </section>  
  );  
}
```

Src/component/TodoList.jsx



Download the React DevTools for a better development experience.

▶ {todo: {...}}

CREATE 호출

```
2 ▶ Uncaught Error: When called with an action of type
  an action, you must explicitly return the previous
    at combination (redux.js:564:1)
    at k (<anonymous>:2235:16)
    at D (<anonymous>:2251:13)
    at <anonymous>:2464:20
    at Object.dispatch (redux.js:288:1)
    at e (<anonymous>:2494:20)
    at onClick (TodoList.js:18:1)
    at HTMLUnknownElement.callCallback (react-dom.development
    at Object.invokeGuardedCallbackDev (react-dom.development
    at invokeGuardedCallback (react-dom.development
```



Reducer

CREATE 구현



Reducer 의 CREATE 동작 구현

- 이제 들어온 Action Type 에 따른 reducer 의 실제 동작을 구현해 봅시다!
- 먼저 CREATE 부터 구현을 해봅시다!
- 혹시 모를 다른 초기 값이 있을지 모르므로 state 를 전개 연산자로 먼저 리턴해 줍니다.
- List 의 경우는 새롭게 입력 받은 값을 list 의 배열에 넣어 주면 됩니다!



```
export default function todo(state = initState, action) {  
  switch (action.type) {  
    case CREATE:  
      return {  
        ...state,  
        list: state.list.concat({  
          id: action.payload.id,  
          text: action.payload.text,  
          done: false,  
        }),  
        nextID: action.payload.id + 1,  
      };  
    case DONE:  
      return {  
        ...state,  
      };  
    default:  
      return state;  
  }  
}
```

Push 말고 Concat 을 사용하는 이유는?

지금은 list 라는 배열에 변경 된 값을
리턴해 줘야 하는 상황입니다!

Push 는 배열에 값을 추가하고 배열의
길이를 리턴해 주고, concat 은 값이 추
가된 배열을 리턴해 줍니다!

따라서 push 를 쓰면 list 에는 숫자 값
만 들어가므로 문제가 생깁니다!

Src/store/modules/todo.js



Dispatch 로 CREATE 호출



CREATE 호출

- 이번에는 CREATE 의 함수에 제대로 된 인자를 전달하여 정상적으로 기능이 작동하도록 해봅시다!
- 리듀서에서 할 일 목록 추가로 필요한 정보는 id 값과 새롭게 추가될 할 일의 text 값이 필요합니다 → 두 데이터를 객체에 담아서 인자로 전달해 봅시다!



```
import { useRef } from "react";
import { useDispatch, useSelector } from "react-redux";
import { create, done } from '../store/modules/todo';

export default function TodoList() {
  const list = useSelector((state) => state.todo.list);
  const inputRef = useRef();
  const dispatch = useDispatch();

  return (
    <section>
      <h1>할일 목록</h1>
      <div>
        <input type="text" ref={inputRef} />
        <button
          onClick={() => {
            dispatch(create({ id: list.length, text: inputRef.current.value }));
            inputRef.current.value = "";
          }}
        >
          추가
        </button>
      </div>
    </section>
  );
}
```

Src/component/TodoList.jsx

할일 목록



추가

- 리액트 공부하기
- 척추의 요정이 말합니다 : 척추 펴기!
- 취업 하기
- 추가가 되나요!?



Reducer

DONE 구현



Reducer 의 DONE 동작 구현

- 동일하게 list 이외의 초기 state 값은 그대로 전달이 되어야 하므로 전개 연산자를 사용!
- List 의 경우는 컴포넌트에서 전달 받은 id 값과 동일한 객체를 찾은 다음 해당 객체의 done 항목을 true 로 변경하면 됩니다!
- 이럴 때는 `map()` 을 쓰면 편합니다! `map()` 은 배열의 모든 값을 순회 하면서 배열의 값을 return 된 값으로 변경해 줍니다!



```
case DONE:
  return {
    ...state,
    list: state.list.map((el) => {
      if (el.id === action.id) {
        return {
          ...el,
          done: true,
        };
      } else {
        return el;
      }
    }),
  };
default:
  return state;
```

Src/store/modules/todo.js



```
// 액션 타입 정의하기
const CREATE = 'todo/CREATE';
const DONE = 'todo/DONE';

// 액션 생성 함수 작성
export function create(payload) {
  return {
    type: CREATE,
    payload,
  };
}

export function done(id) {
  return {
    type: DONE,
    id,
  };
}

// 초기 상태 설정
const initState = {
  list: [
    {
      id: 0,
      text: '리액트 공부하기',
      done: false,
    },
    {
      id: 1,
      text: '척추의 요정이 말합니다 : 척추 펴기!',
      done: false,
    },
    {
      id: 2,
      text: '취업 하기',
      done: false,
    },
  ],
};
```

```
// 리듀서 설정(실제 작업은 이친구가 합니다!)
export default function todo(state = initState, action) {
  switch (action.type) {
    case CREATE:
      return {
        ...state,
        list: state.list.concat({
          id: action.payload.id,
          text: action.payload.text,
          done: false,
        }),
        nextID: action.payload.id + 1,
      };
    case DONE:
      return {
        ...state,
        list: state.list.map((el) => {
          if (el.id === action.id) {
            return {
              ...el,
              done: true,
            };
          } else {
            return el;
          }
        }),
      };
    default:
      return state;
  }
}
```

Src/store/modules/todo.js

전체 코드



Dispatch 로

DONE 호출



DONE 호출

- 이번에는 DONE 의 함수에 제대로 된 인자를 전달하여 정상적으로 기능이 작동하도록 해봅시다!
- 리듀서에서 완료 된 목록의 id 값만 받아서 해당 목록의 done 항목을 true 로 변경만 하면 됩니다!
- Done 함수에 인자로 id 값을 전달해 봅시다!



```
<ul>
  {list.map((el) => {
    return (
      <li key={el.id}>
        {el.text}
        <button
          onClick={() => {
            dispatch(done(el.id));
          }}
        >
          완료
        </button>
      </li>
    );
  })}
</ul>
```

Src/component/ToDoList.js



각각 컴포넌트에 Filter 걸기!



Filter 처리!

- 지금 `<TodoList>` 컴포넌트와 `<DoneList>` 컴포넌트는 동일한 List 를 출력하고 있습니다!
- 이제 done 의 값을 통해 필터링 하여 `<TodoList>` 에는 할 일 목록만, `<DoneList>` 에는 완료 된 목록만 남겨 봅시다!



TodoList 컴포넌트

- <TodoList> 컴포넌트 List 의 항목 중에서 done 의 값이 false 인 친구들만 가져오면 됩니다!
- 배열의 filter 메소드는 조건식을 만족하는 배열만 남겨서 리턴해 주므로 해당 메소드를 사용 하면 됩니다!

```
export default function TodoList() {  
  const list = useSelector((state) => state.todo.list).filter(  
    (el) => el.done === false  
  );  
};
```

Src/component/TodoList.jsx





DoneList 컴포넌트

- **<DoneList>** 컴포넌트 List 의 항목 중에서 done 의 값이 true 인 친구들만 가져오면 됩니다!

```
export default function DoneList() {  
  const list = useSelector((state) => state.todo.list).filter(  
    (el) => el.done === true  
  );  
}
```

Src/component/DoneList.js





하지만
언제나 조져지는 건
나였다



킹치만...

- 역시 테스트를 해보니 에러가 뜨네요!

```
✖ Warning: Encountered two children with the same key, react-dom.development.js:86  
`0`. Keys should be unique so that components maintain their identity across  
updates. Non-unique keys may cause children to be duplicated and/or omitted – the  
behavior is unsupported and could change in a future version.  
  at ul  
  at section  
  at DoneList (http://localhost:3000/static/js/bundle.js:103:72)  
  at ListContainer  
  at div  
  at App  
  at Provider (http://localhost:3000/static/js/bundle.js:37892:5)
```




원인은...

- List 요소의 key 값은 고유해야 하지만 고유하지 않아서 생기는 문제입니다!
- TodoList 에서 할 일을 추가 할 때, Store 에서 받아온 list 의 length 값을 넘기고 있습니다 → 이미 완료를 몇 개 하면 list 의 길이가 짧아짐 → 새로 생성 되는 요소는 이전의 key 와 동일한 값을 가지게 됨 → 에러 발생!



해결책은...

- 할 일 목록의 id를 목록의 순번으로 부여를 하고 있으므로, 해당 순번도 store 에서 전역으로 관리하여 문제를 막아 봅시다!
- 이런 부분이 리얼 redux 실전 입니다! 😊



문제 해결하기!



ID 를 관리하기 위한 State 생성!

- Store 의 todo 모듈에 ID 관리를 위한 값을 설정해 봅시다!
- 일단 초기 List 의 길이 값을 구하고, 해당 값을 다음에 생성 될 할 일 목록의 ID 값으로 넘겨주는 구조를 그려 봅시다!



```
const initState = {  
  // 초기 상태 설정  
};  
  
let counts = initState.list.length;  
initState['nextID'] = counts;
```

Src/store/modules/todo.js



CREATE 리듀서에 해당 내용 추가!

- CREATE 액션이 호출 되면 nextID 의 값을 새로운 할 일 목록의 id 로 전달 되기 때문에, 그 다음에 CREATE 가 호출 되기 전에 nextID 값은 +1 상태가 되어야 합니다!
- 따라서, action 에서 받아온 id 값(이전 상태의 nextID 값)에 +1 을 해주면 됩니다!

```
export default function todo(state = initState, action) {  
  switch (action.type) {  
    case CREATE:  
      return {  
        ...state,  
        list: state.list.concat({  
          id: action.payload.id,  
          text: action.payload.text,  
          done: false,  
        }),  
        nextID: action.payload.id + 1,  
      };  
  }  
}
```





할 일 목록 추가 시 기능 수정



<TodoList> 컴포넌트 기능 수정

- 이제는 새롭게 만들어질 할 일 목록의 id 는 list.length 로 보내는 것이 아니라 Store 의 **todo.js** 모듈에서 받아오면 됩니다!
- 해당 값을 CREATE 액션 호출 시 전달해 주면, 리듀서에서 그 값을 받아 다음 할 일 목록의 id 값을 +1 시켜 주므로 논리적으로 문제 없이 구성이 가능합니다!

```
export default function TodoList() {
  const list = useSelector((state) => state.todo.list).filter(
    (el) => el.done === false
  );

  const nextID = useSelector((state) => state.todo.nextID);

  const inputRef = useRef();
  const dispatch = useDispatch();

  return (
    <section>
      <h1>할일 목록</h1>
      <div>
        <input type="text" ref={inputRef} />
        <button
          onClick={() => {
            dispatch(create({ id: nextID, text: inputRef.current.value }));
            inputRef.current.value = '';
          }}
        >
          추가
        </button>
      </div>
    </section>
  );
}
```





개발자 MBTI 조사



개발자가 흔히 접하는 상황에 따라서 MBTI 를
알아 봅시다!

테스트 시작

퇴근 직전에 동료로부터 개발자 모임에 초대를
받은 나!



퇴근 시간에 나는?

그런 모임을 왜 이제서야 알려 준거야! 당장 모
임으로 출발한다

VS

1년 전에 알려줬어도 안갔을 건데 뭘... 더 빠르
게 집으로 간다

1 / 4

서비스 출시 이틀 전 야근 시간, 갑자기 동료가
어!? 를 외쳤다!

나의 선택은?

무슨 버그가 발생한 거지? 아마 DB 관련 버그
가 아닐까? 빠르게 동료의 자리로 달려간다

VS

아... 내일도 야근 각이구나 ㅠㅠ! 일단 동료의 자
리로 가 본다

3 / 4

당신의 개발자 MBTI 결과는?



자유로운 영혼으로 개발팀의 윤희유 및 활력소
가 되어줄 당신의 MBTI 는!

ENFP

이건 재미로 읽어 보세요!

외교형
4

ENFP

- 재기발랄한 활동가 -

좋은일컴퍼니^주

- 긍정적이며 낙천적임. 인싸인 경우가 많음
- 친구들과 잘 어울리고 다른 사람들과 같이 있는 것을 좋아함
- 새로운 인간관계에 두려움이 없음
- 순간 집중력이 좋아서 벼락치기 해도 성과가 잘 나옴
- 그러나 끈기가 없어 반복적인 일상을 매우 극혐함
- 감정이 풍부하고 그 감정이 표정에서 다 드러남
- 관종끼가 강하지만 의외로 내향성이 강하고 독립적인 편
- 계획 세우기 귀찮아함, 즉흥적임



<https://dev-mbti.tetz.org/>



이걸 통해 무엇을 배우나요!?

- 리액트 SPA(Single Page Application) 제작
- Styled-Components 활용
 - 글로벌 스타일 적용
 - 컴포넌트 디자인
- Redux 활용



기초 세팅!



기초 세팅!


- 먼저 새롭게 만들 app 을 만들어 봅시다!
 - `Npx create-react-app mbti-app`
- 필요 모듈을 한큐에 설치!
 - `npm i redux react-redux @reduxjs/toolkit styled-components`



폴더 구조 세팅

폴더 구조 세팅!

- Redux 활용을 위한 폴더 구조를 만들어 봅시다!
- src
 - components
 - store
 - modules
 - mbts.js
 - Index.js



```
> .vscode
> node_modules
v public
  > images
  ★ favicon.ico
  <> index.html
  🖼 logo192.png
  🖼 logo512.png
  {} manifest.json
  ≡ robots.txt
v src
  > components
  v store
    v modules
      JS mbti.js 1
      JS index.js
  # App.css
```



Redux 세팅!



Redux 기초 세팅!

- Src 폴더의 최상위 index.js 파일 세팅
- Vscode 의 추천 대로 createStore 가 아닌 configureStore 사용!
- rootReducer 임포트
- Provider 임포트 후, App 감싸주기 + store 부여
- Redux 개발자 도구 사용을 위한 코드 추가!



```
import { configureStore } from '@reduxjs/toolkit';
import rootReducer from './store';
import { Provider } from 'react-redux';

const reduxDevTool =
  window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__();

const store = configureStore({ reducer: rootReducer }, reduxDevTool);
```

Src/index.js

- configureStore 는 rootReducer 를 객체 형태로 전달!



```
import { configureStore } from '@reduxjs/toolkit';
import rootReducer from './store';
import { Provider } from 'react-redux';

const reduxDevTool =
  window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__();

const store = configureStore({ reducer: rootReducer }, reduxDevTool);
console.log(store.getState());

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

Src/index.js

- Provider 컴포넌트로 App 컴포넌트 감싸기 + store 설정



rootReducer

설정



rootReducer 설정

- `Src/store/index.js` 에서 선언된 리듀서를 임포트 하고 있으므로 해당 파일로 이동!
- 사실상 SPA(Single Page App) 이기 때문에 `combineReducers` 를 활용 할 필요가 없지만, 나중에 위해 연습!

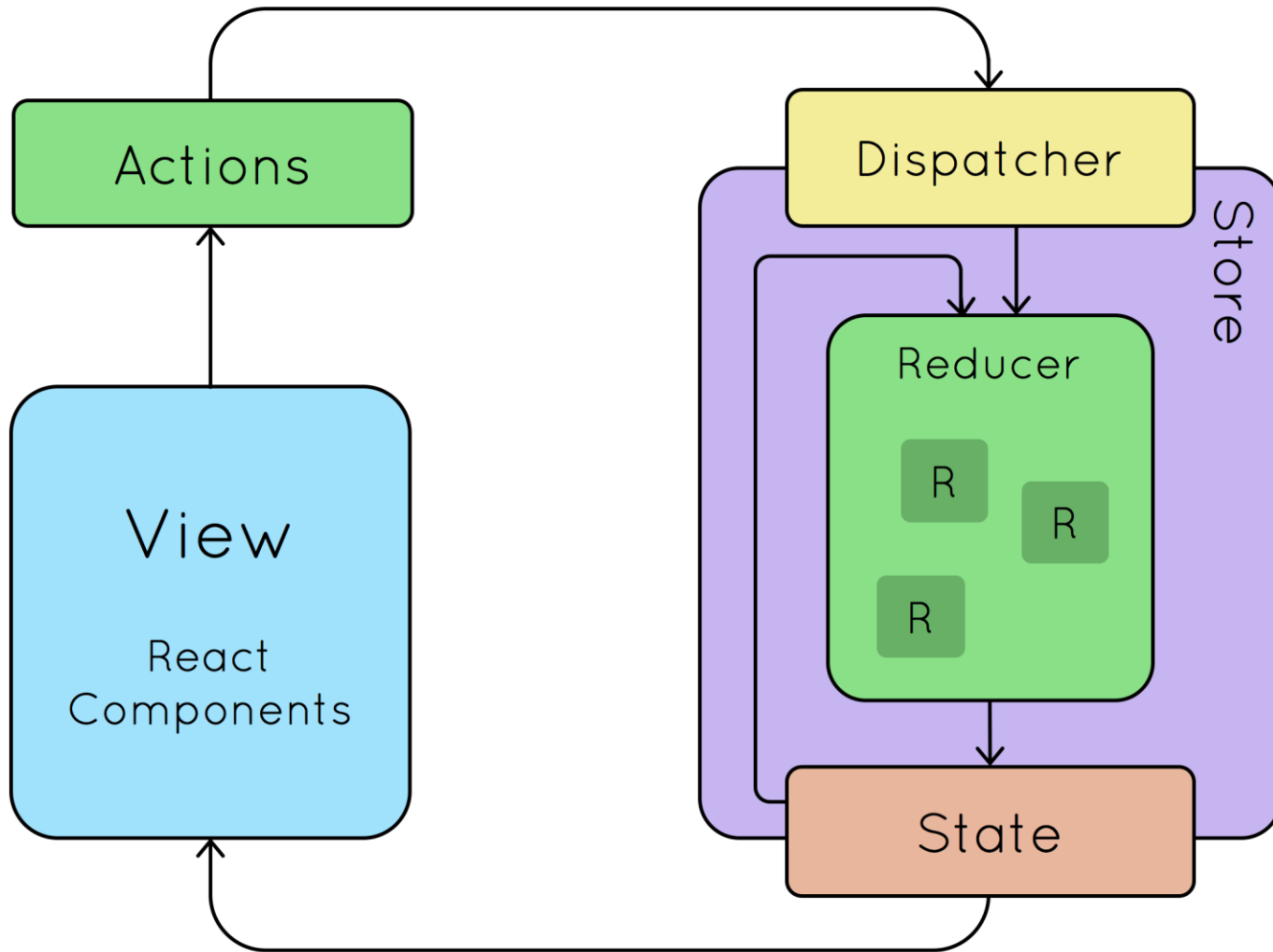
```
import { combineReducers } from 'redux';  
import mbti from '../modules/mbti';  
  
export default combineReducers({  
  mbti,  
});
```

Src/store/index.js



- 추후 모듈이 추가 되었을 때, 이런 구조를 만들어 놓으면 편리 합니다!
- 새로운 SPA 가 추가될 경우 보통 Redux 모듈이 추가되고, 해당 SPA 는 라우팅으로 구현을 합니다!

Redux 동작 순서





mbti store

설정



mbti store 설정

- 실제로 일을 하게 될, mbti store 를 설정해 봅시다!
- 초기 State 를 설정
- DB 연동을 하지 않을 것이므로 필요 데이터 설정!
- 액션 타입 설정
- 액션 함수 설정
- 리듀서 만들기!



초기 상태 설정



초기 상태 설정

- MBTI 질문 목록
- 현재 페이지 값
- Mbti 전체 결과 값
- 전체 결과에 대한 설명 값
 - 추가 이미지 주소 값


```
// 초기 상태 설정
const initState = {
  mbtiResult: '',
  page: 0, // 0: 인트로 페이지, 1 ~ n: 선택 페이지, n+1: 결과 페이지
  survey: '질문 목록',
  explanation: '결과에 대한 설명'
};
```

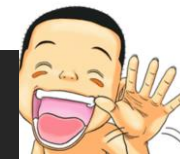
Src/store/modules/mbti.js





```
survey: [
  {
    question:
      '퇴근 직전에 동료로부터 개발자 모임에 초대를 받은 나!\n\n퇴근 시간에 나는?',
    answer: [
      {
        text: '그런 모임을 왜 이제서야 알려 준거야! 당장 모임으로 출발한다',
        result: 'E',
      },
      {
        text: '1년 전에 알려줬어도 안갔을 건데 뭐... 더 빠르게 집으로 간다',
        result: 'I',
      },
    ],
  },
  {
    question:
      '새로운 서비스 개발 중에, 동료가 새로 나온 신기술을 쓰는게 더 편할거라고 추천을 해준다!\n\n나의 선택은!?',
    answer: [
      {
        text: '원소리며, 그냥 하던 대로 개발하면 되는거지! 기존 생각대로 개발한다',
        result: 'S',
      },
      {
        text: '오호? 그런게 있어? 일단 구글을 찾아본다',
        result: 'N',
      },
    ],
  },
  {
    question:
      '서비스 출시 이틀 전 야근 시간, 갑자기 동료가 어!? 를 외쳤다!\n\n나의 선택은?',
    answer: [
      {
        text: '무슨 버그가 발생한 거지? 아마 DB 관련 버그가 아닐까? 빠르게 동료의 자리로 달려간다',
        result: 'T',
      },
      {
        text: '아... 내일도 야근 각이구나  $\pi\pi$ ! 일단 동료의 자리로 가 본다',
        result: 'F',
      },
    ],
  },
  {
    question:
      '팀장님이 xx씨 그전에 말한 기능 내일 오후까지 완료 부탁드립니다라고 말했다!\n\n나의 선택은?',
    answer: [
      {
        text: '일단 빠르게 개발 완료하고, 나머지 시간에 놀다',
        result: 'J',
      },
      {
        text: '그거 내일 아침에 와서 개발해도 충분 하겠는데? 일단 놀다',
        result: 'P',
      },
    ],
  },
],
```

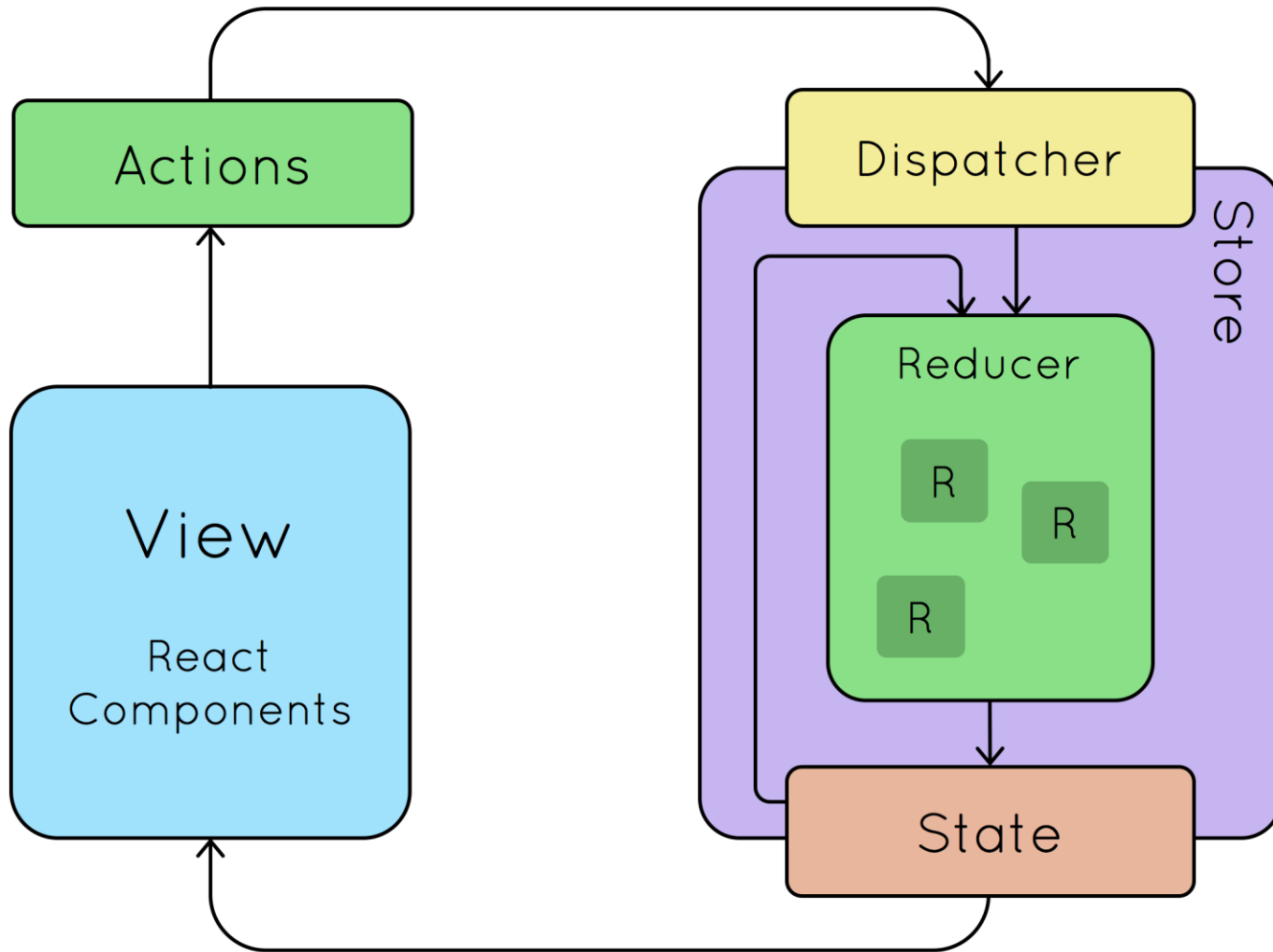
Src/store/modules/mbti.js 중
질문 항목



```
explanation: {
  ESTJ: {
    text: '무리한 개발 일정만 아니라면 일정을 철저하게 지킬 당신의 MBTI 는!',
    img: '/images/estj.jpg',
  },
  ISTJ: {
    text: '스스로 하고싶은 분야를 끝까지 파고 들어서 끝내 성공 시킬 당신의 MBTI 는!',
    img: '/images/istj.jpg',
  },
  ENTJ: {
    text: '미래의 능력 켜는 개발 팀장님으로 개발팀을 이끌 당신의 MBTI 는!',
    img: '/images/entj.jpg',
  },
  INTJ: {
    text: '혼자서 모든 것을 다 해내는 원맨 캐리의 표본! 당신의 MBTI 는!',
    img: '/images/intj.jpg',
  },
  ESFJ: {
    text: '개발팀의 분위기 메이커이자 아이디어 뱅크가 될 당신의 MBTI 는!',
    img: '/images/esfj.jpg',
  },
  ISFJ: {
    text: '개발팀의 마더 테레사, 고민 상담소 역할을 자처하는 당신의 MBTI 는!',
    img: '/images/isfj.jpg',
  },
  ENFJ: {
    text: '당신이 있는 팀은 언제나 올바른 곳을 향하고 있습니다! 팀원은 물론 팀의 방향을 챙기는 당신의 MBTI 는!',
    img: '/images/enfj.jpg',
  },
  INFJ: {
    text: '예리한 통찰력으로 모든 것을 내다보면서 완벽하게 개발을 할 당신의 MBTI 는!',
    img: '/images/infj.jpg',
  },
  ESTP: {
    text: '쿨하게 자신이 할 것을 하면서 논리적인 개발을 할 당신의 MBTI 는!',
    img: '/images/estp.jpg',
  },
  ISTP: {
    text: '단시간에도 효율적으로 개발하여 모든 것을 완성할 당신의 MBTI 는!',
    img: '/images/istp.jpg',
  },
  ENTP: {
    text: '스스로 흥미만 생긴다면 당장에 페이스북도 만들어 버릴 당신의 MBTI 는!',
    img: '/images/entp.jpg',
  },
  INTP: {
    text: '확실한 주관과 뛰어난 지능을 바탕으로 논리적 개발을 할 당신의 MBTI 는!',
    img: '/images/intp.jpg',
  },
  ESFP: {
    text: '개발팀의 에너지아저! 개발팀 특유의 서먹함을 깨는 당신! 당신의 MBTI 는!',
    img: '/images/esfp.jpg',
  },
  ISFP: {
    text: '뛰어난 호기심과 예술적 감각으로 개발팀의 부족함을 채워줄 당신! 당신의 MBTI 는!',
    img: '/images/isfp.jpg',
  },
  ENFP: {
    text: '자유로운 영혼으로 개발팀의 윤활유 및 활력소가 되어줄 당신의 MBTI 는!',
    img: '/images/enfp.jpg',
  },
  INFP: {
    text: '개발팀의 그 어떤 트러블도 당신 앞에서는 사르르 녹을뿐, 팀의 근간을 다져주는 당신의 MBTI 는!',
    img: '/images/infp.jpg',
  },
},
```

Src/store/modules/mbti.js 중
결과 설명 항목

Redux 동작 순서





Action Type 설정



Action Type 설정

- 지금 App 에서 필요한 Action Type 은 어떤 것들이 있을까요?
- 먼저 페이지를 다음 장으로 넘기는 기능!
 - 전달 값? → 필요 X
- 선택에 따른 결과를 반영하는 기능!
 - 전달 값? → 선택에 따른 결과 값 전달 필요
- 마지막 페이지에서 결과를 리셋하는 기능!
 - 전달 값? → 필요 X



```
// 액션 타입(문자열)
const CHECK = 'mbti/CHECK';
const NEXT = 'mbti/NEXT';
const RESET = 'mbti/RESET';
```

Src/store/modules/mbti.js



Action

생성 함수 설정



Action 생성 함수 설정

- 외부에서 Store 내부 함수의 구조는 알 필요는 없습니다!
- 외부에서 원하는 Action 에 따른 기능을 Dispatch 를 통해 전달할 Action 함수를 설정해 봅시다!
- 지금 있는 Action Type 은 CHECK / NEXT / RESET 이므로 각각 Type 에 맞는 함수를 설정해 봅시다!



```
// 액션 생성 함수
// payload -> 선택에 다른 결과 값 result 전달 필요
export function check(result) {
  return {
    type: CHECK,
    payload: { result },
  };
}

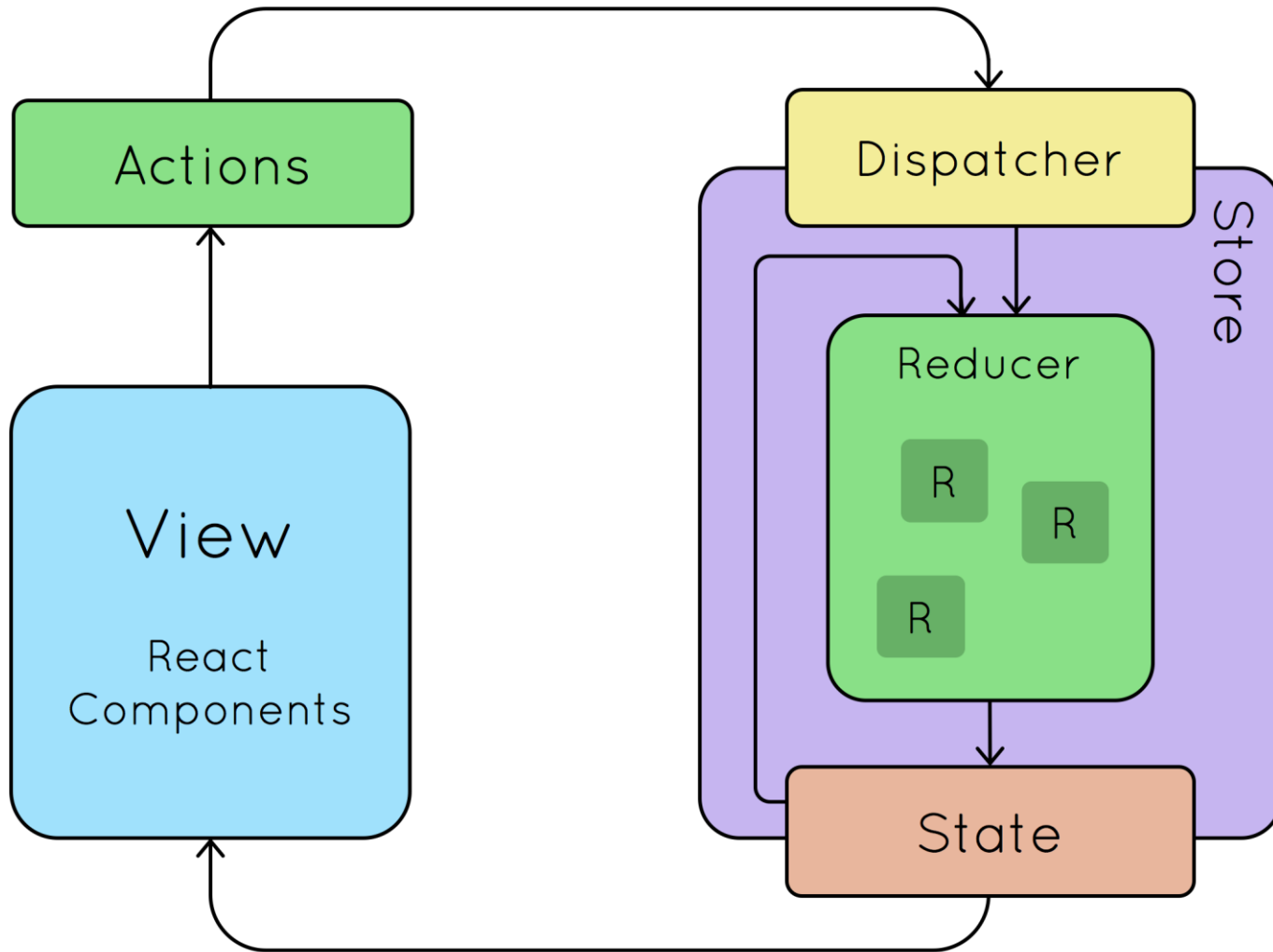
export function next() {
  return {
    type: NEXT,
  };
}

export function reset() {
  return {
    type: RESET,
  };
}
```

- 외부에서 사용해야 하므로 export 설정
- Type은 반드시 전달 필요
- 데이터가 필요한 경우 payload 에 담아서 전달

Src/store/modules/mbti.js

Redux 동작 순서





Reducer

만들기



Reducer 만들기

- 이제 실제로 **State** 변경 관리하는 **Reducer** 를 만들어 봅시다!
- **Dispatch** 에 의해 전달 받은 action 의 type 값에 따라 원하는 기능을 수행하는 역할을 합니다
- **Reducer** 가 해당 파일의 export default 가 됩니다!
- Type 구분은 보통 Switch 를 통해 사용합니다
 - 코드 가독성이 if 문 대비 좋고, Default 가 강제 되는 부분!



// 리뷰서

```
export default function mbti(state = initState, action) {  
  switch (action.type) {  
    case CHECK:  
      return {  
        ...state,  
        mbtiResult: state.mbtiResult + action.payload.result,  
      };  
    case NEXT:  
      return {  
        ...state,  
        page: state.page + 1,  
      };  
    case RESET:  
      return {  
        ...state,  
        page: 0,  
        mbtiResult: '',  
      };  
    default:  
      return state;  
  }  
}
```

- mbtiResult 값은 조사 항목에 있는 result 의 문자열을 순서대로 더 하면 되므로 + 연산자 사용
- 초기 State 에 다른 값이 있을 수 있으므로 전개 연산자로 나머지 값 전달

Src/store/modules/mbti.js



// 리뷰서

```
export default function mbti(state = initState, action) {  
  switch (action.type) {  
    case CHECK:  
      return {  
        ...state,  
        mbtiResult: state.mbtiResult + action.payload.result,  
      };  
    case NEXT:  
      return {  
        ...state,  
        page: state.page + 1,  
      };  
    case RESET:  
      return {  
        ...state,  
        page: 0,  
        mbtiResult: '',  
      };  
    default:  
      return state;  
  }  
}
```

- 단순히 page 의 값을 + 1 시켜 주면 끝!
- 초기 State 에 다른 값이 있을 수 있으므로 전개 연산자로 나머지 값 전달

Src/store/modules/mbti.js



// 리뷰서

```
export default function mbti(state = initState, action) {  
  switch (action.type) {  
    case CHECK:  
      return {  
        ...state,  
        mbtiResult: state.mbtiResult + action.payload.result,  
      };  
    case NEXT:  
      return {  
        ...state,  
        page: state.page + 1,  
      };  
    case RESET:  
      return {  
        ...state,  
        page: 0,  
        mbtiResult: '',  
      };  
    default:  
      return state;  
  }  
}
```

- 결과 값을 초기화 하고, page 를 0 으로 만들어 주면 끝!
- 초기 State 에 다른 값이 있을 수 있으므로 전개 연산자로 나머지 값 전달

Src/store/modules/mbti.js



컴포넌트 제작 기초 작업!



App.js

코드 정리



App.js 코드 정리

- React 기본 코드를 정리해 봅시다!
- 하는 김에 public 폴더의 index.html 의 주석도 정리 합시다!

```
function App() {  
  return (  
    <>  
    <h1>APP 페이지 입니다!</h1>  
    </>  
  );  
}
```

```
export default App;
```

Src/App.js





시작 페이지

제작



개발자 MBTI 조사



개발자가 흔히 접하는 상황에 따라서 MBTI 를
알아 봅시다!

테스트 시작



시작 페이지 제작(Start.jsx)

- 페이지가 로딩 되면 제일 처음 보이는 **Start 페이지**를 제작해 봅시다!
- 글자와 이미지, 버튼의 조합으로 간단하게 만들어 봅시다!
- 버튼은 페이지 리로딩을 막기 위해 `<a>` 태그로 구현!
- **Styled 컴포넌트**를 사용하여 꾸미기!
- 컴포넌트와 분리 하기 위해 pages 폴더를 만들고 만들기!



```
import styled from 'styled-components';

export default function Start() {
  return (
    <>
      <p>개발자 MBTI 조사</p>
      
      <p>개발자가 흔히 접하는 상황에 따라서 MBTI 를 알아 봅시다!</p>
      <a text="테스트 시작">테스트 시작</a>
    </>
  );
}
```

Src/pages/Start.jsx



Styled-components 꾸미기!

- Styled-components 를 사용해서 꾸며 봅시다!
- 각각의 태그를 별도로 이름을 변경하고, 해당 컴포넌트를 변수로 받은 다음
Styled-components 를 사용하여 디자인!

```
export default function Start() {
  return (
    <>
      <Header>개발자 MBTI 조사</Header>
      <MainImg src="/images/main.jpg" alt="메인 이미지" />
      <SubHeader>
        개발자가 흔히 접하는 상황에 따라서 MBTI 를 알아 봅시다!
      </SubHeader>
      <a text="테스트 시작">테스트 시작</a>
    </>
  );
}
```

Src/pages/Start.jsx



```
const MainImg = styled.img`  
  width: inherit;  
`;  
  
const Header = styled.p`  
  font-size: 3em;  
`;  
  
const SubHeader = styled.p`  
  font-size: 1.5em;  
  color: #777;  
`;  
;
```

Src/pages/Start.jsx





시작 페이지

삽입 및 확인



Start 페이지 삽입 + 메인 틀 잡기

- 작성한 Start 페이지를 Main 컴포넌트에 삽입
- 전체 컴포넌트를 담을 컨테이너 역할을 하는 Main 컴포넌트의 스타일도 잡아 줍시다!



```
const Main = styled.main`  
  box-sizing: border-box;  
  width: 100%;  
  max-width: 500px;  
  padding: 0 35px;  
  margin: auto;  
  text-align: center;  
`;  
;
```

```
function App() {  
  return (  
    <>  
      <Main>  
        <Start />  
      </Main>  
    </>  
  );  
}
```

```
export default App;
```

Src/App.js



Button

컴포넌트 제작



Button 컴포넌트 제작

- Button 컴포넌트는 테스트 시작, MBTI 선택지 선택, 다시 하기 등등 다양한 곳에서 재사용이 될 예정입니다!
- React 의 특수화 개념을 사용해서 기초 스타일인 Button 컴포넌트를 제작하고 해당 컴포넌트를 이용하여 각각의 색과 기능을 가진 버튼으로 만들어 사용해 봅시다!

Button 컴포넌트 제작

테스트 시작



- Button 컴포넌트는 props로 부터 받아와야 할 값이 버튼의 텍스트, 이벤트 핸들러, 메인 색상, 서브 색상, Hover 시 색상의 값을 받아와야 합니다!
- MyButton 이라고 명명한 이후, Styled-components 로 꾸미기
- Styled-components 는 현재 컴포넌트에서 전달한 props 를 받아서 처리가 가능하므로 편리하게 랜더링 시점에 디자인이 결정 되는 다이나믹 디자인이 가능



```
import styled from 'styled-components';

export default function Button({
  text,
  clickEvent,
  mainColor,
  subColor,
  hoverColor,
}) {
  return (
    <MyButton
      onClick={clickEvent}
      mainColor={mainColor}
      subColor={subColor}
      hoverColor={hoverColor}
    >
      {text}
    </MyButton>
  );
}
```

- Styled-components 에 props 를 전달하기 위한 props 전달!

Src/component/Button.js



```
const MyButton = styled.a`
  position: relative;
  display: inline-block;
  cursor: pointer;
  vertical-align: middle;
  text-decoration: none;
  line-height: 1.6em;
  font-size: 1.2em;
  padding: 1.25em 2em;
  background-color: ${(props) => props.mainColor};
  border: 2px solid ${(props) => props.subColor};
  border-radius: 0.75em;
  user-select: none;
  transition: transform 0.15s ease-out;
  transform-style: preserve-3d;
  margin-top: 1em;
```

- 전달 받은 props 의 사용
- Props 를 인자로 받아서 Styled 에 적용이 가능!

Src/component/Button.js



```
&::before {
  content: '';
  position: absolute;
  width: 100%;
  height: 100%;
  top: 0;
  right: 0;
  left: 0;
  right: 0;
  background: ${({props}) => props.subColor};
  border-radius: inherit;
  box-shadow: 0 0 0 2px ${({props}) => props.subColor};
  transform: translate3d(0, 0.75em, -1em);
}
&:hover {
  background: ${({props}) => props.hoverColor};
  transform: translateY(0.25em);
}
;
```

- SASS 와 마찬가지로 & 를 사용해 스스로 지칭 가능!
- 가상 요소, 클래스 선택자 사용 가능

Src/component/Button.js

```
import styled from 'styled-components';
```

```
const MyButton = styled.a`
  position: relative;
  display: inline-block;
  cursor: pointer;
  vertical-align: middle;
  text-decoration: none;
  line-height: 1.6em;
  font-size: 1.2em;
  padding: 1.25em 2em;
  background-color: ${(props) => props.mainColor};
  border: 2px solid ${(props) => props.subColor};
  border-radius: 0.75em;
  user-select: none;
  transition: transform 0.15s ease-out;
  transform-style: preserve-3d;
  margin-top: 1em;
  &::before {
    content: '';
    position: absolute;
    width: 100%;
    height: 100%;
    top: 0;
    right: 0;
    left: 0;
    right: 0;
    background: ${(props) => props.subColor};
    border-radius: inherit;
    box-shadow: 0 0 0 2px ${(props) => props.subColor};
    transform: translate3d(0, 0.75em, -1em);
    transition: transform 0.15s ease-out;
  }
  &:hover {
    background: ${(props) => props.hoverColor};
    transform: translateY(0.25em);
  }
`;
```

```
export default function Button({
  text,
  clickEvent,
  mainColor,
  subColor,
  hoverColor,
}) {
  return (
    <MyButton
      onClick={clickEvent}
      mainColor={mainColor}
      subColor={subColor}
      hoverColor={hoverColor}
    >
      {text}
    </MyButton>
  );
};
```

Src/component/Button.js

전체 코드





OrangeButton

으로 특수화!



OrangeButton 으로 특수화

- 기본이 되는 **Button 컴포넌트**를 만들었으므로 특수화를 사용하여 **OrangeButton 컴포넌트**를 제작해 봅시다!
- 원하는 텍스트와 색상 값을 **props** 로 전달하고, 이벤트 핸들러는 사용 시점에서 결정이 될 것이므로 그대로 전달만 합시다!



```
import Button from './Button';

export default function OrangeButton({ text, clickEvent }) {
  return (
    <Button
      text={text}
      clickEvent={clickEvent}
      mainColor="#fae243"
      subColor="#fa9f1a"
      hoverColor="#faf000"
    />
  );
}
```

Src/component/OrangeButton.js



OrangeButton

적용!



OrangeButton 적용

- Start 컴포넌트에 OrangeButton 을 적용하여 봅시다!

```
import OrangeButton from './OrangeButton';

export default function Start() {
  return (
    <>
      <Header>개발자 MBTI 조사</Header>
      <MainImg src="/images/main.jpg" alt="메인 이미지" />
      <SubHeader>
        개발자가 흔히 접하는 상황에 따라서 MBTI 를 알아 봅시다!
      </SubHeader>
      <OrangeButton text="테스트 시작" />
    </>
  );
}
```

Src/component/Start.js



Styled-components

GlobalStyle



글로벌 스타일 적용!

- 리엑트는 다양한 컴포넌트의 조합으로 사용이 됩니다!
- 그래서 보통 컴포넌트 단위로 디자인이 적용이 되죠
- 그런데, 페이지 전체에 대한 폰트 또는 기본 스타일이 필요하다면 어떻게 하면 될까요?



글로벌 스타일 적용!

- SPA 인 경우는 App.css 에 글로벌 스타일을 적용하면 되고
- MPA 인 경우에는 전체를 감싸는 최종 컴포넌트에 스타일을 적용 해도 됩니다!
- 다만, Styled-components 의 경우는 이러한 방식보다 자체 기능을 통해 전체 페이지에 글로벌 스타일을 적용합니다!



GlobalStyle 컴포넌트 제작하기

- Components 폴더에 GlobalStyle.js 파일을 만들고 **GlobalStyle 컴포넌트**를 제작해 봅시다!
- Styled-components 는 **createGlobalStyle** 이라는 메소드를 제공하여 글로벌 스타일 적용을 가능하게 합니다!
- 필요한 것들을 설정해 봅시다!



```
import { createGlobalStyle } from 'styled-components';

const GlobalStyle = createGlobalStyle`
  @font-face {
    font-family: 'ONE-Mobile-POP';
    src: url('https://cdn.jsdelivr.net/gh/projectnoonnu/noonfonts_2105_2@1.0/ONE-Mobile-POP.woff') format('woff');
    font-weight: normal;
    font-style: normal;
  }

  body {
    font-family: 'ONE-Mobile-POP', "Arial", sans-serif;
    padding-top: 1em;
    white-space: pre-wrap;
  }

  ul, ol {
    list-style: none;
    padding-left: 0px;
  }
`;
export default GlobalStyle;
```

Src/component/GlobalStyle.js



GlobalStyle 적용하기

- 만든 GlobalStyle 컴포넌트를 적용하고자 하는 App 의 최상단에 컴포넌트로 넣어주면 Global Style 이 적용이 됩니다!



```
import GlobalStyle from './components/GlobalStyle';
```

```
function App() {  
  return (  
    <>  
      <GlobalStyle />  
      <Main>  
        <Start />  
      </Main>  
    </>  
  );  
}
```

```
export default App;
```

Src/App.js

개발자 MBTI 조사



개발자가 흔히 접하는 상황에 따라서 MBTI 를
알아 봅시다!

테스트 시작



페이지 분기 처리



페이지 분기 처리!

- 지금 만드는 App 은 페이지에 따라서 보여줘야 하는 부분이 다릅니다!
- Page 가 0 이면 → Start 페이지 보여주기
- Page 가 설문의 길이와 같이 같을 때 까지 → 설문 조사 페이지 보여주기
- Page 가 설문의 길이를 넘어가면 → 결과 페이지 보여주기



페이지 분기 처리!

- Page 의 상태에 따라서 각각의 페이지를 렌더링 하는 방식으로 분기 처리를 해봅시다!
- 리액트의 경우 라우팅 보다는 조건부 렌더링 또는 3항 연산자, if 문으로 처리 해주는 방법이 더 편리합니다!



```
import { useSelector } from 'react-redux';
import styled from 'styled-components';
import GlobalStyle from './components/GlobalStyle';
import Start from './components/Start';

function App() {
  const page = useSelector((state) => state.mbti.page);

  return (
    <>
      <GlobalStyle />
      <Main>
        {page === 0 ? <Start /> : <Mbti />}
      </Main>
    </>
  );
}

export default App;
```

- Store 의 page 값이 0 이면 Start 페이지를, 아닐 경우 Mbti 조사를 하는 Mbti 페이지를 보여주기

Src/App.js



Mbti 페이지 제작



퇴근 직전에 동료로부터 개발자 모임에 초대를
받은 나!

퇴근 시간에 나는?

그런 모임을 왜 이제서야 알려 준거야! 당장 모
임으로 출발한다

VS

1년 전에 알려줬어도 안갔을 건데 뭐... 더 빠르
게 집으로 간다

1 / 4





Mbti 페이지 제작하기

- 이제 설문을 하는 MbtI 페이지를 제작해 봅시다!
- 기존의 Button 컴포넌트를 활용해서 설문을 선택하는 SkyblueButton 컴포넌트를 작성하고 활용!
- Page 의 번호를 가져와서 해당 번호에 맞는 설문문의 text 값을 SkyblueButton 에 담아서 출력 하기!



SkyblueButton

특수화



SkyblueButton 특수화

- 기존 OrangeButton 을 활용하여 색상 값만 변경하여 Skyblue 버튼 만들기!



```
import Button from './Button';

export default function SkyblueButton({ text, clickEvent }) {
  return (
    <Button
      text={text}
      clickEvent={clickEvent}
      mainColor="#7EDCFA"
      subColor="#3A82E0"
      hoverColor="#CFECEF2"
    />
  );
}
```

Src/component/SkyblueButton.js



페이지 제작

```
import { useSelector } from 'react-redux';
import styled from 'styled-components';
import SkyblueButton from './SkyblueButton';
```

```
export default function Mbti() {
  const survey = useSelector((state) => state.mbti.survey);
  const page = useSelector((state) => state.mbti.page);

  return (
    <>
      <SurveyQuestion>{survey[page - 1].question}</SurveyQuestion>
      <ul>
        {survey[page - 1].answer.map((el, index) => {
          return (
            <li key={index}>
              <SkyblueButton
                text={el.text}
              />
              {index === 0 && <Vs>VS</Vs>}
            </li>
          );
        })}
      </ul>
    </>
  );
}
```

- Store 에서 값 받아오기

- 설문 항목의 index 는 0 부터 시작이므로 page-1 의 인덱스로 접근하여 설문 항목 가져오기
- 선택지는 배열에 담겨 있으므로 map 메소드를 이용하여 각각의 버튼을 그려주기
- Vs 는 처음에 한번만 그려지면 되므로 map 의 index 를 이용하여 조건부 렌더링 처리

```
const SurveyQuestion = styled.p`  
  font-size: 1.5em;  
  color: #777;  
`;  
;
```

```
const Vs = styled.p`  
  font-size: 2em;  
  padding-top: 1em;  
`;  
;
```

- 컴포넌트 디자인

Src/pages/Mbti.js





이벤트 핸들러에 액션 생성 함수 지정



액션 생성 함수 지정

- 이제 페이지를 넘기는 기능을 하는 액션 생성 함수인 `next()` 를 `dispatch` 를 이용하여 Reducer 에 전달해 봅시다!
- Start 컴포넌트의 테스트 시작이라는 버튼에 지정!
- Mbti 컴포넌트의 선택지 선택 버튼에 지정!



```
import { useDispatch } from 'react-redux';  
import { next } from "../store/modules/mbti";
```

```
export default function Start() {  
  const dispatch = useDispatch();
```

- useDispatch 혹은 dispatch 지정
- 테스트 시작 버튼 클릭 시, next() 액션 생성 함수를 dispatch 로 reducer 에 전달

```
  return (  
    <>
```

```
    <Header>개발자 MBTI 조사</Header>
```

```
    <MainImg src="/images/main.jpg" alt="메인 이미지" />
```

```
    <SubHeader>
```

```
      개발자가 흔히 접하는 상황에 따라서 MBTI 를 알아 봅시다!
```

```
    </SubHeader>
```

```
    <OrangeButton text="테스트 시작" clickEvent={() => dispatch(next())} />
```

```
  </>
```

```
);
```

```
}
```

Src/pages/Start.js

```
import { useDispatch } from 'react-redux';
import { next } from "../store/modules/mbti";
```

Src/pages/Mbti.js



```
export default function Mbti() {
  const survey = useSelector((state) => state.mbti.survey);
  const page = useSelector((state) => state.mbti.page);
  const dispatch = useDispatch();
  return (
    <>
      <SurveyQuestion>{survey[page - 1].question}</SurveyQuestion>
      <ul>
        {survey[page - 1].answer.map((el, index) => {
          return (
            <li key={index}>
              <SkyblueButton
                text={el.text}
                clickEvent={() => { dispatch(next()) }}
              />
              {index === 0 && <Vs>VS</Vs>}
            </li>
          );
        })}
      </ul>
    </>
  );
}
```

- 설문 선택 시, next() 액션 생성 함수를 dispatch 로 reducer 에 전달



Progress Bar

만들기



Progress Bar 만들기

- 현 진행 상황을 보여주는 Progress Bar 도 만들어 봅시다!
- Progress.js 컴포넌트 작성하기
- Progress Bar 의 값은 현재 Page / 전체 설문 배열의 길이 값으로 표시하면 됩니다!
- Progress Bar 의 바깥 부분을 먼저 그리고, 자식 요소가 부모의 크기를 상속 한 다음 색을 입혀서 % 로 구현



```
import styled from 'styled-components';

export default function Progress({ page, maxPage }) {
  return (
    <MyProgress>
      <div>
        {page} / {maxPage}
      </div>
      <Fill>
        <Gauge percent={ (page / maxPage) * 100 }></Gauge>
      </Fill>
    </MyProgress>
  );
}
```

- % 로 게이지를 그릴 것이므로 %에 들어갈 숫자 값을 계산하여 전달

Src/component/Progress.js



```
const MyProgress = styled.div`
  margin-top: 3em;
`;

const Fill = styled.div`
  width: 100%;
  height: 10px;
  background-color: #777;
  margin-top: 1em;
  text-align: left;
`;

const Gauge = styled.div`
  background-color: skyblue;
  display: inline-block;
  height: inherit;
  position: relative;
  top: -4px;
  width: ${(props) => props.percent}%
`;
```

- Props 로 값을 전달 받아 게이지를 그려서 상황에 따라 처리

Src/component/Progress.js



Progress Bar

삽입



Progress Bar 삽입

- 만들어진 ProgressBar 를 삽입해 줍시다
- Props 로 전달할 값은, 현재 page 값과 전체 설문의 수 이므로 해당 정보도 props 로 전달해 주면 됩니다!



```
export default function Mbti() {
  const survey = useSelector((state) => state.mbti.survey);
  const page = useSelector((state) => state.mbti.page);
  const dispatch = useDispatch();

  return (
    <>
      <SurveyQuestion>{survey[page - 1].question}</SurveyQuestion>
      <ul>
        {survey[page - 1].answer.map((el, index) => {
          return (
            <li key={index}>
              <SkyblueButton
                text={el.text}
                clickEvent={() => {
                  dispatch(next());
                }}
              />
              {index === 0 && <Vs>VS</Vs>}
            </li>
          );
        })}
      </ul>
      <Progress page={page} maxPage={survey.length} />
    </>
  );
}
```

- Progress 컴포넌트를 삽입하고 필요한 props 값도 전달!



결과를 만드는
Check() 삽입!



Check() 액션 생성 함수 삽입!

- 이제 페이지는 잘 넘어 갑니다!
- 그럼 MBTI 조사 결과 값을 만들어야 겠죠!?
- 해당 기능은 CHECK Action 이 담당합니다!



Check() 액션 생성 함수

```
// payload -> 선택에 다른 결과 값 result 전달 필요
export function check(result) {
  return {
    type: CHECK,
    payload: { result },
  };
}
```

Src/store/modules/mbti.js

- Check() 액션 생성 함수는 결과 값 만을 전달 받네요?



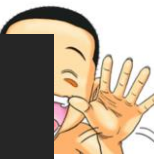
리듀서에서의 동작

```
case CHECK:
  return {
    ...state,
    mbtiResult: state.mbtiResult + action.payload.result,
  };

```

Src/store/modules/mbti.js

- 액션 생성 함수로 전달 받은 결과를 mbtiResult 라는 결과 문자열에 추가를 해주는 액션이 끝입니다!
- 그럼, check() 함수를 호출 할 때, 설문 객체에 포함 된 결과 문자열만 전달 하면 되겠군요!



```
survey: [  
  {  
    question:  
      '퇴근 직전에 동료로부터 개발자 모임에 초대를 받은 나!\n\n퇴근 시간에 나는?',  
    answer: [  
      {  
        text: '그런 모임을 왜 이제서야 알려 준거야! 당장 모임으로 출발한다',  
        result: 'E',  
      },  
      {  
        text: '1년 전에 알려줬어도 안갔을 건데 뭘... 더 빠르게 집으로 간다',  
        result: 'I',  
      },  
    ],  
  },  
],  
},
```

- 
- 이 Result 값만 전달 하면 됩니다!



Dispatch 로
Check() 전달!



```
return (  
  <>  
    <SurveyQuestion>{survey[page - 1].question}</SurveyQuestion>  
    <ul>  
      {survey[page - 1].answer.map((el, index) => {  
        return (  
          <li key={index}>  
            <SkyblueButton  
              text={el.text}  
              clickEvent={() => {  
                dispatch(check(el.result));  
                dispatch(next());  
              }}  
            />  
            {index === 0 && <Vs>VS</Vs>}  
          </li>  
        );  
      })}  
    </ul>  
    <Progress page={page} maxPage={survey.length} />  
  </>  
)
```

- 설문 항목에 있던 결과 문자열의 값을 check()의 인자로 전달!

Src/component/Mbti.js

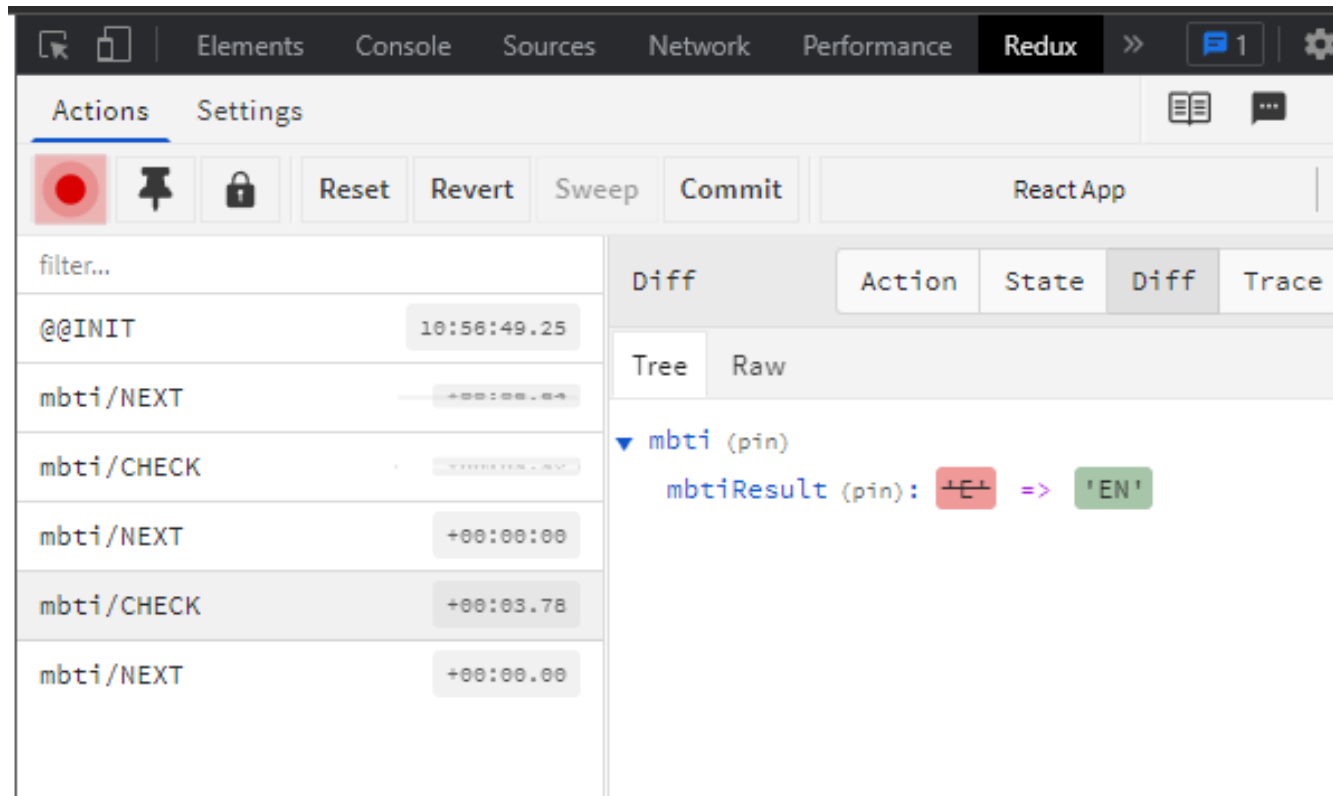


동작 확인



실제 동작이 잘 되는지 확인해 봅시다!

- 이럴 때 저번에 설치 해 두었던 Redux Devtools 가 역할을 합니다!



```
▼ mbti (pin)  
  mbtiResult (pin): 'EN' => 'ENFJ'
```



결과 출력

페이지 작성



SkyblueButton

특수화



PinkButton 특수화

- 기존 OrangeButton 을 활용하여 색상 값만 변경하여 PinkButton 만들기!



```
import Button from './Button';

export default function SkyblueButton({ text, clickEvent }) {
  return (
    <Button
      text={text}
      clickEvent={clickEvent}
      mainColor="#7EDCFA"
      subColor="#3A82E0"
      hoverColor="#CFECF2"
    />
  );
}
```

Src/component/SkyblueButton.js



이제 redux 에 모인 결과를 출력

- MBTI 결과가 잘 반영 되는 것을 확인 하였으니, 해당 결과를 보여줄 결과 페이지인 Show.jsx 를 만들어 봅시다!
- **Show 페이지**는 Mbti 최종 결과가 들어있는 mbtiResult 값과, Mbti 결과 값에 맞는 설명 + 이미지를 출력해 주면 됩니다!
- 먼저 텍스트 부터 입력해서 디자인 부터 하고 결과 값 출력을 해봅시다!



```
import styled from 'styled-components';

export default function Show() {
  return (
    <>
      <Header>당신의 개발자 MBTI 결과는?</Header>
      <Explanation>결과 설명 출력</Explanation>
      <Result>결과 출력</Result>
      <Additional>이건 재미로 읽어 보세요!</Additional>
      <AdditionalImg src={} alt="팩폭" />
      <OrangeButton text="다시 검사하기" clickEvent={} />
    </>
  );
}
```

Src/component/Show.js

```
const Header = styled.p`  
  font-size: 3em;  
`;  
;
```

```
const Explanation = styled.p`  
  font-size: 1.5em;  
  color: #777;  
`;  
;
```

```
const Result = styled.p`  
  font-size: 3em;  
  color: dodgerblue;  
`;  
;
```

```
const Additional = styled.p`  
  font-size: 2em;  
  color: orange;  
`;  
;
```

```
const AdditionalImg = styled.img`  
  width: 500px;  
  transform: translateX(-35px);  
`;  
;
```

Src/component/Show.js





Redux 에서 결과 값 출력하기!



Redux 에서 결과 값 받아서 출력하기!

- 페이지 디자인은 마쳤으니 이제 Redux 에서 결과 값을 받아서 출력해 봅시다!
- MBTI 결과는 Store 의 mbtiResult 에 있으므로 해당 값을 받아오기
- 그리고 설명은 각각의 MBTI 결과 값을 Key 로 가지는 객체로 선언을 하였기 때문에 편리하게 접근이 가능합니다!



```
export default function Show() {  
  const result = useSelector((state) => state.mbti.mbtiResult);  
  const explanation = useSelector((state) => state.mbti.explanation[result]);  
  const dispatch = useDispatch();  
  
  return (  
    <>  
      <Header>당신의 개발자 MBTI 결과는?</Header>  
      <Explanation>{explanation.text}</Explanation>  
      <Result>{result}</Result>  
      <Additional>이건 재미로 읽어 보세요!</Additional>  
      <AdditionalImg src={explanation.img} alt="팩폭" />  
      <PinkButton text="다시 검사하기" clickEvent={} />  
    </>  
  );  
}
```

Src/component/Show.js



Reset()

액션 생성 함수 전달



다시하기 버튼 기능 추가!

- 이제 다시하기 버튼을 눌렀을 때, 페이지가 최초로 돌아가는 기능을 추가해 주면 됩니다!
- 이미 RESET 기능(page 를 0 으로 만들고,mbtiResult 를 초기화)은 구현이 되었으니 dispatch 를 통해 전달만 합시다!



```
import { useDispatch, useSelector } from "react-redux";
import styled from "styled-components";
import PinkButton from "../components/PinkButton";
import { reset } from "../store/modules/mbti";

export default function Show() {
  const result = useSelector((state) => state.mbti.mbtiResult);
  const explanation = useSelector((state) => state.mbti.explanation[result]);
  const dispatch = useDispatch();

  return (
    <>
      <Header>당신의 개발자 MBTI 결과는?</Header>
      <Explanation>{explanation.text}</Explanation>
      <Result>{result}</Result>
      <Additional>이건 재미로 읽어 보세요!</Additional>
      <AdditionalImg src={explanation.img} alt="팩폭" />
      <PinkButton text="다시 검사하기" clickEvent={() => dispatch(reset())} />
    </>
  );
}
```

Src/pages/Show.jsx



App.js

분기 처리!



App.js 분기 처리

- 이제 결과 페이지도 보여줘야 하기 때문에, 결과 페이지에 대한 분기 처리도 해봅시다!
- Page 가 0 → Start 컴포넌트
- Page 가 1 ~ n → Mbti 컴포넌트
- Page 가 n + 1 → Show 컴포넌트
- If 문을 쓰는 것 보다는 간단하게 3항 연산자를 2중으로 써서 처리 해봅시다!



```
function App() {  
  const page = useSelector((state) => state.mbti.page);  
  const survey = useSelector((state) => state.mbti.survey);  
  
  return (  
    <>  
      <GlobalStyle />  
      <Main>  
        {page === 0 ? (  
          <Start />  
        ) : page !== survey.length + 1 ? (  
          <Mbti />  
        ) : (  
          <Show />  
        )}  
      </Main>  
    </>  
  );  
}
```

Src/App.js





수고하셨습니다!