



with



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 윤활유 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 prettier`

Prettier 세팅



```
{  
  "semi": true,  
  "singleQuote": true  
}
```

/.prettierrc

```
{  
  "[javascript]": {  
    "editor.formatOnSave": true,  
    "editor.defaultFormatter": "esbenp.prettier-vscode"  
  }  
}
```


/.vscode/settings.json



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';

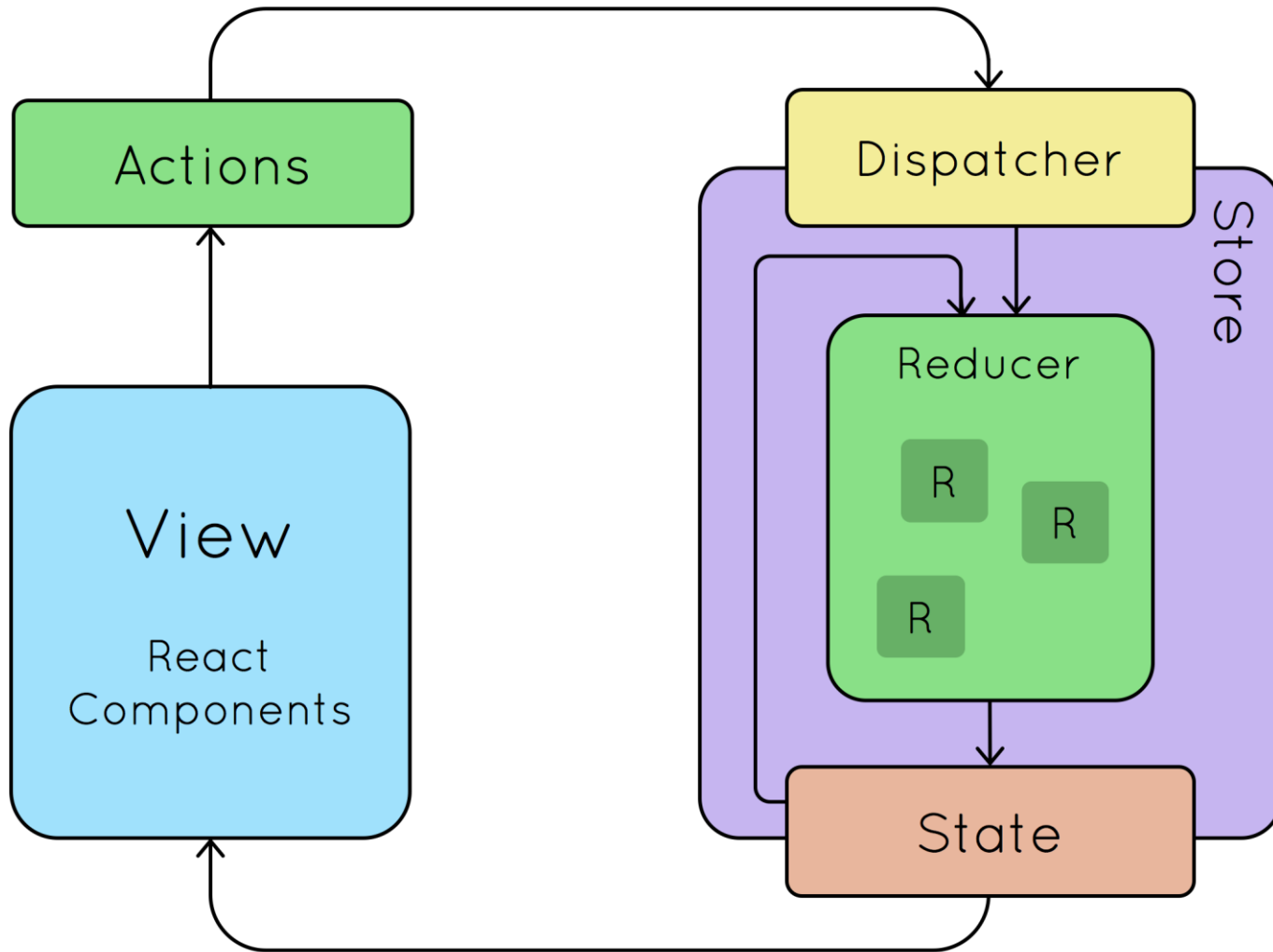
export default combineReducers({
  mbti,
});
```

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'
};
```

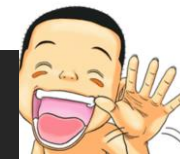
Src/store/modules/mbti.js





```
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: '원소리여, 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 ㅠㅠ! 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',
      },
    ],
  },
],
```

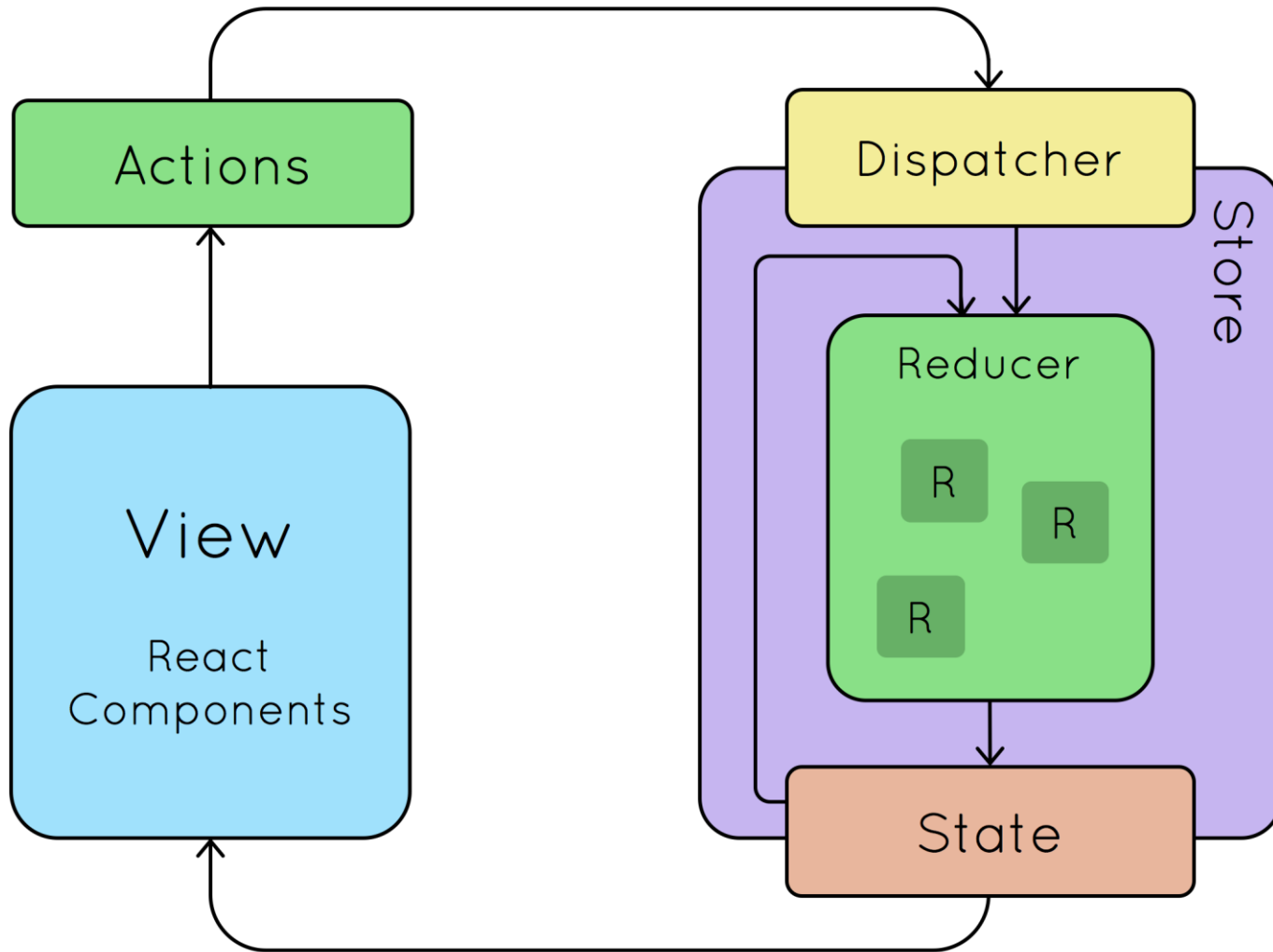
Src/store/modules/mbti.js 중
질문 항목



```
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 에너지이저! 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 },
  };
}

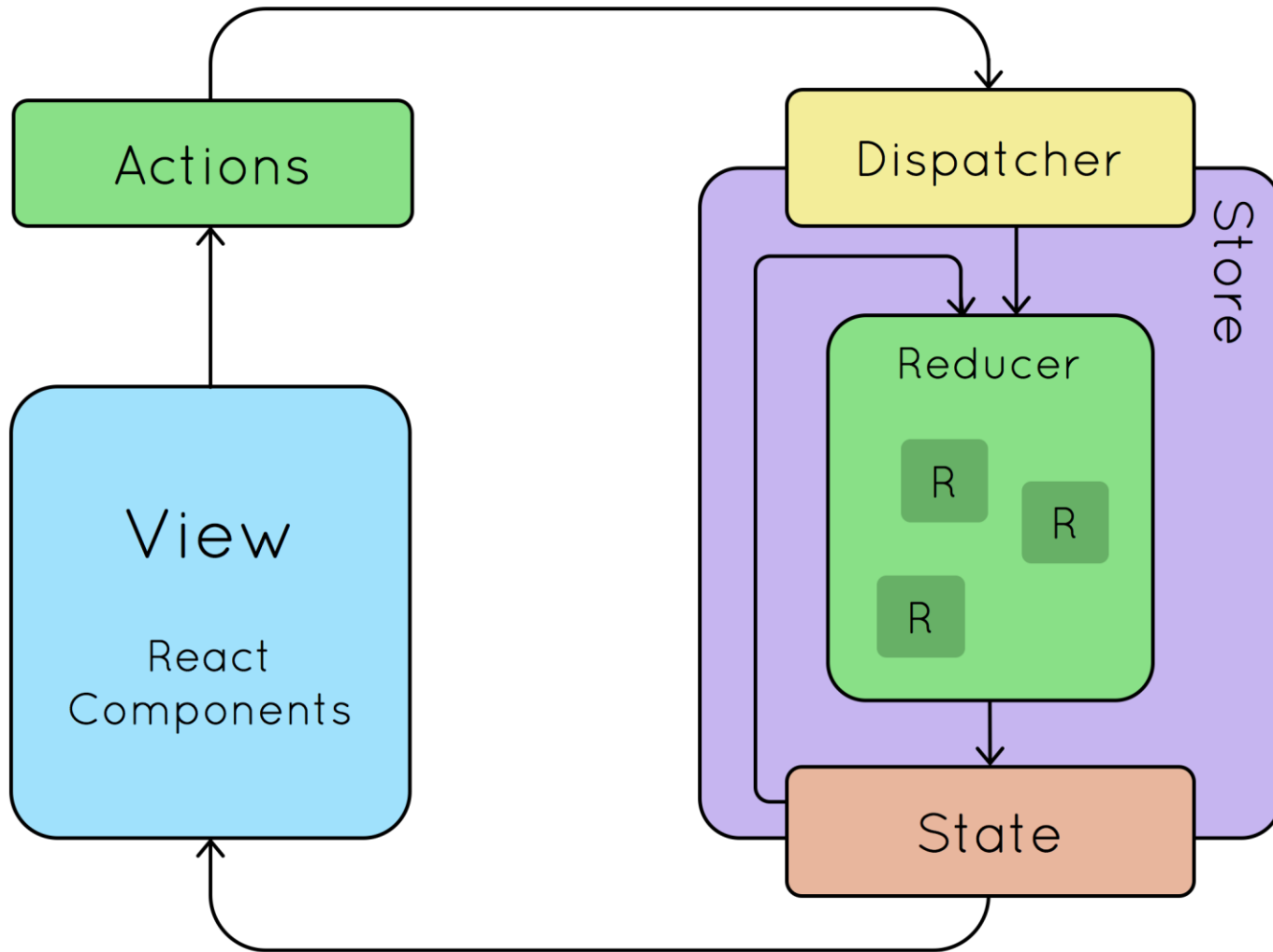
export function next() {
  return {
    type: NEXT,
  };
}

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 설문의 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="#CFECF2"
    />
  );
}
```

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} />
    </>
  );
}
```

Src/component/Mbti.js



- 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?



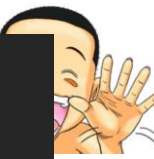
리듀서에서의 동작

```
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

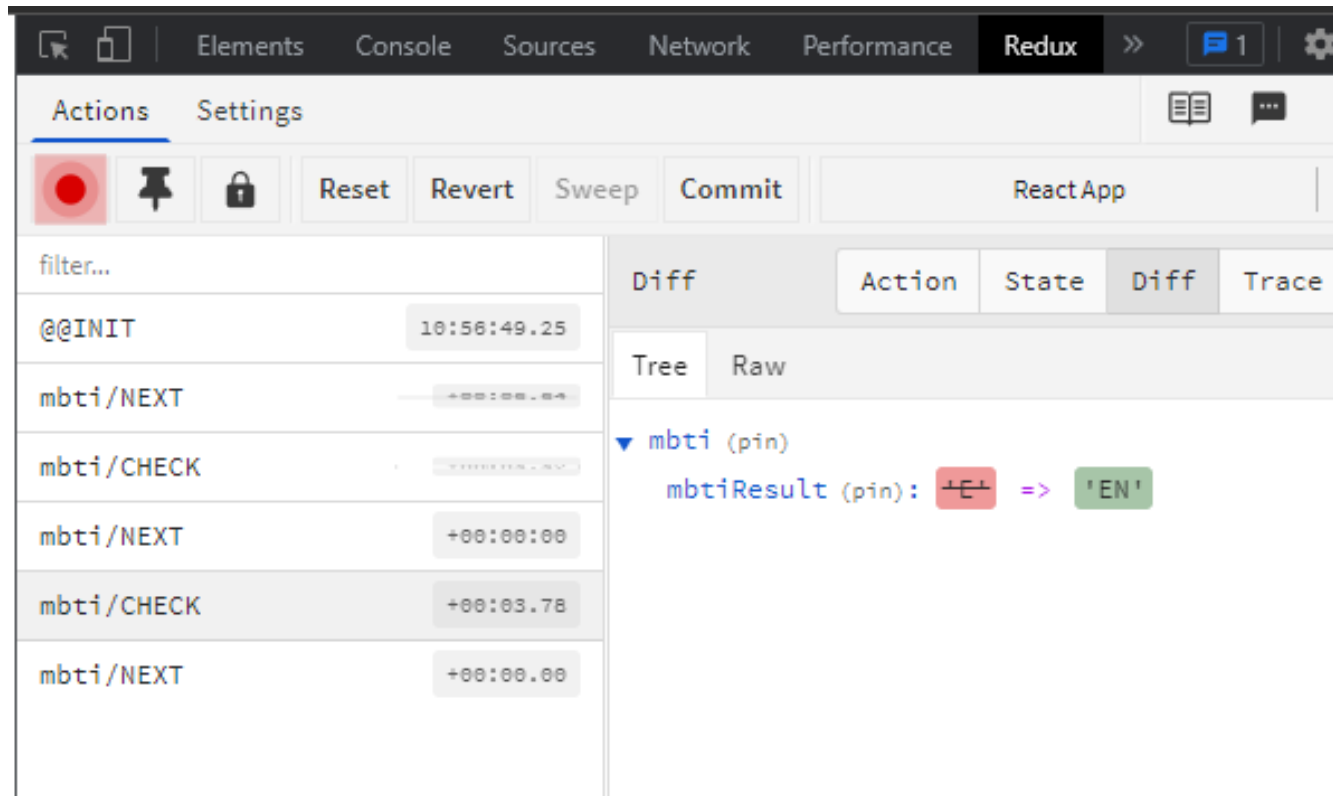


동작 확인



실제 동작이 잘 되는지 확인해 봅시다!

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



```
▼ mbti (pin)  
  mbtiResult (pin): 'ENF' => '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="#CFECEF2"
    />
  );
}
```

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 합니다!



```
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





MongoDB로 구현하기

MongoDB와 React를 같이 사용해 봅시다!



- 이제 State 값을 변수로 쓰는 것이 아니라 DB로 부터 받아와 봅시다!
- 저희가 사용할 데이터는 복잡한 형태의 객체 이므로 MongoDB를 사용하는 편이 더 유리 합니다!

MongoDB와 React를 같이 사용해 봅시다!



- 이제 State 값을 변수로 쓰는 것이 아니라 DB로 부터 받아와 봅시다!
- 저희가 사용할 데이터는 복잡한 형태의 객체 이므로 MongoDB를 사용하는 편이 더 유리 합니다!



간단한 백엔드 서버 구축하기

간단한 백엔드 서버 구성하기!



- 일단 리액트 폴더에 data-server 폴더 생성
- Express 와 cors 모듈 설치
- Npm i express cors mongodb

```
✓ data-server
```



MongoDB 접속 모듈 생성

- mongoConnect.js 에 Mongodb 에 접속하기 위한 모듈을 만들어 봅시다!

```
const { MongoClient, ServerApiVersion } = require('mongodb');

const DB_URI_ATLAS =
  'mongodb+srv://tetz:qwer1234@cluster0.sdiakr0.mongodb.net/?retryWrites=true&w=majority';

const uri = DB_URI_ATLAS;

const client = new MongoClient(uri, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  serverApi: ServerApiVersion.v1,
});

module.exports = client;
```

Data-server/mongoConnect.js



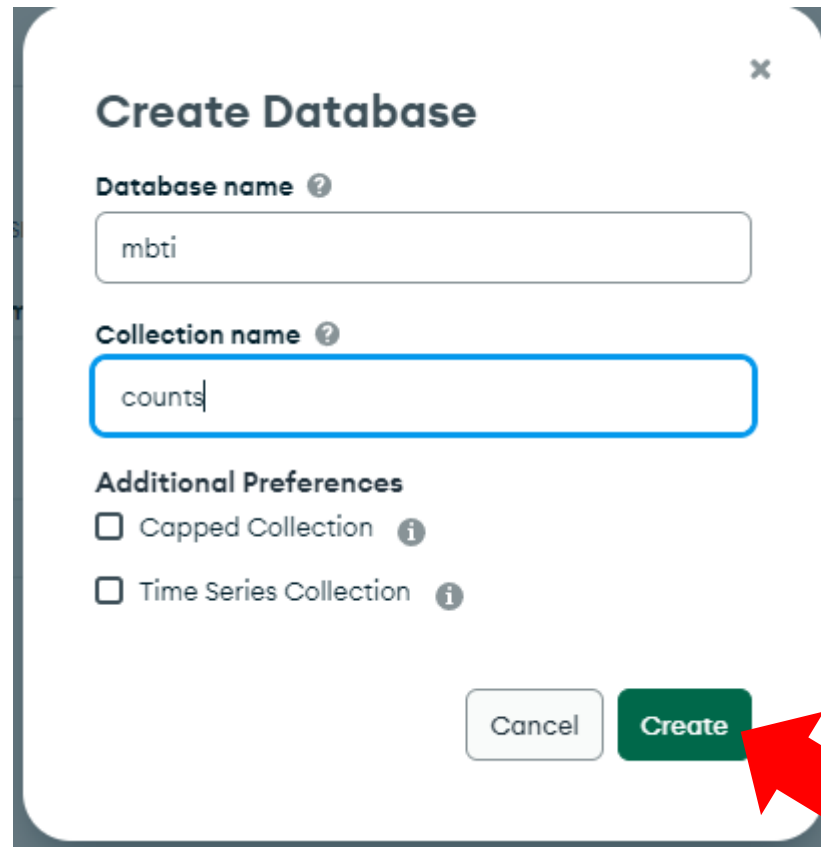
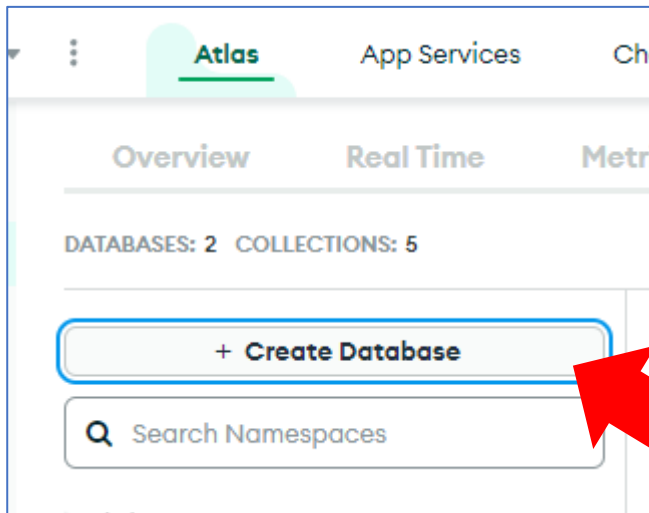
MongoDB

데이터 베이스 만들기!

Atlas 에 접속하여 mbti 용 DB를 만듭시다!



- Atlas 에 접속하여 mbti 용 DB 를 구축해 봅시다!





방문자 확인용 Collection

- 먼저 방문자 수 확인용 collection 에 document 를 추가해 봅시다!

mbti.counts

STORAGE SIZE: 4KB LOGICAL DATA SIZE: 0B TOTAL DOCUMENTS: 0 INDEXES TOTAL SIZE: 4KB

Find

Indexes

Schema Anti-Patterns 0

Aggregation

Search Indexes ●

FILTER

{ field: 'value' }

▶ OPTIMIZE

Apply

Reset

INSERT DOCUMENT

QUERY RESULTS: 0



```
_id: ObjectId('639cd0066a99bf623cb8bec4')  
id: 1  
counts: 0
```



Mbti 설문용 Collection

- 설문용 Data 는 Collection 을 만들고 하나하나 넣으면 될까요?
- 아니죠! 우리는 Redux 에 객체 형태로 데이터를 가지고 있으므로, 해당 데이터를 mogodb 의 insert 기능을 이용하여 바로 넣어 주면 됩니다!
- 테이블을 별도로 만드는 과정을 할 필요가 없고, JS의 객체와 mongodb 의 데이터가 바로 연동 되는 부분이 편리한 점 입니다! 😊
 - 물론 단점도 존재 합니다!



MongoDB

Controller 생성




MongoDB 용 Controller 만들기!

- Mongodb 에서 데이터를 받아서 처리해줄 Controller 도 만들어 봅시다
- 먼저, 설문용 데이터를 DB에 저장 시켜주는 controller 부터 만들어 봅시다!
- 설문용 데이터는 Redux 에서 객체를 바로 가져 오면 됩니다!



```
// 초기 상태 설정
const initState = {
  mbtiResult: '',
  page: 0, // 0: 인트로 페이지, 1 ~ n: 선택 페이지, n+1: 결과 페이지
  survey: [ ...
],
  explanation: { ...
},
};
```

kdtTetz, 14시간 전 • 수업 코드 수정 Data-server/controllers/mongoController.js



```
const mongoClient = require('../mongoConnect');
const _client = mongoClient.connect();

const mongoDB = {
  setData: async () => {
    const client = await _client;
    // collection 이 없으면 알아서 생성
    const db = client.db('mbti').collection('data');
    // redux 의 데이터를 그대로 mongodb 에 삽입
    const result = await db.insertOne(initState);
    if (result.acknowledged) {
      return '업데이트 성공';
    } else {
      throw new Error('통신 이상');
    }
  },
};
```

```
module.exports = mongoDB;
```

Data-server/controllers/mongoController.js



MongoDB 용 라우터 설정



MongoDB 용 라우터 설정

- 이제 mongoDB 에 데이터를 요청하는 주소 라우팅 설정을 해봅시다!
- 메인 서버에 **/mongo** 라는 주소 값을 처리하는 라우터 넣어주기
- Mongo.js 라는 라우터 모듈 생성



메인 서버 설정

```
const express = require('express');
const cors = require('cors');
const PORT = 4000;

const app = express();

app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cors());

const mongoRouter = require('./routes/mongo');
app.use('/mongo', mongoRouter);

app.listen(PORT, () => {
  console.log(`데이터 통신 서버가 ${PORT}에서 작동 중`);
});
```

Data-server/server.js



Mongo.js 라우터 구현

```
const express = require('express');
const router = express.Router();

const mongoDB = require('../controllers/mongoController');

router.post('/setdata', async (req, res) => {
  const msg = await mongoDB.setData();
  res.send(JSON.stringify(msg));
});

module.exports = router;
```

Data-server/routes/mongo.js



POSTMAN으로 요청 보내기



POSTMAN 으로 요청 보내기

- 이제 POST 방식으로 <http://localhost:4000/mongo/setdata> 에 요청을 보내 봅시다!



HomeWorkspaces ▾API Network ▾Explore

My Workspace

NewImport

GET localhost:4000/users ●POST localhost:4000/users ●PUT localhost:4000/users

Collections

APIs

Environments

Mock Servers

Monitors

Flows

History

+ ▾

New Collection

GET localhost:4000/users

POST localhost:4000/users

PUT localhost:4000/users

DEL localhost:4000/users

POST localhost:4000/posts

PUT localhost:4000/posts

DEL localhost:4000/posts

New Collection / localhost:4000/posts

POST ▾localhost:4000/mongo/setdata

ParamsAuthorizationHeaders (9)BodyPre-request ScriptTes

Query Params

KEY

Key

BodyCookiesHeaders (8)Test Results

PrettyRawPreviewVisualizeHTML ▾

1"업데이트 성공"



Overview

Real Time

Metrics

Collections

DATABASES: 2 COLLECTIONS: 5

+ Create Database

Q Search Namespaces

▶ kdt1

▼ mbti

counts

data

mbti.data

STORAGE SIZE: 20KB

Find

Indexe

FILTER

{ field: 'va

QUERY RESULTS: 1-1 OF 1

```
_id: ObjectId('634be9aa42bdff49301db252')
mbtiResult: ""
page: 0
▼ survey: Array
  ▼ 0: Object
    question: "퇴근 직전에 동료로부터 개발자 모임에 초대를 받은 나!

              퇴근 시간에 나는?"
  ▼ answer: Array
    ▼ 0: Object
      text: "그런 모임을 왜 이제서야 알려 준거야! 당장 모임으로 출발한다"
      result: "E"
    ▼ 1: Object
      text: "1년 전에 알려줬어도 안갔을 건데 뭐... 더 빠르게 집으로 간다"
      result: "I"
  > 1: Object
  > 2: Object
  > 3: Object
> explanation: Object
```

QUERY RESULTS: 1-1 OF 1

```
_id: ObjectId('634be9aa42bdff49301db252')
mbtiResult: ""
page: 0
> survey: Array
> explanation: Object
```




컨트롤러 기능 추가



컨트롤러 기능 추가!

- 이제 기존 MySQL 에서 하던 기능들을 컨트롤러에 추가해 봅시다!
- 방문자 수를 가져오는 getCounts
- 방문자 수를 증가 시키는 incCounts
- 데이터를 받아오는 getData

getCounts() 구현



```
getCounts: async () => {  
  const client = await _client;  
  const db = client.db('mbti').collection('counts');  
  const data = await db.find({}).toArray();  
  return data;  
},
```

Data-server/controllers/mongoController.js



incCounts() 구현

```
incCounts: async () => {  
  const client = await _client;  
  const db = client.db('mbti').collection('counts');  
  const result = await db.updateOne({ id: 1 }, { $inc: { counts: +1 } });  
  if (result.acknowledged) {  
    return '업데이트 성공';  
  } else {  
    throw new Error('통신 이상');  
  }  
},
```

Data-server/controllers/mongoController.js

getData() 구현



```
getData: async () => {  
  const client = await _client;  
  const db = client.db('mbti').collection('data');  
  const data = await db.find({}).toArray();  
  return data;  
},
```

Data-server/controllers/mongoController.js



Data-server/controllers/mongoController.js

전체 코드(initState 데이터는 제외)

```
const mongoClient = require('../mongoConnect');
const _client = mongoClient.connect();

const mongoDB = {
  getCounts: async () => {
    const client = await _client;
    const db = client.db('mbti').collection('counts');
    const data = await db.find({}).toArray();
    return data;
  },
  incCounts: async () => {
    const client = await _client;
    const db = client.db('mbti').collection('counts');
    const result = await db.updateOne({ id: 1 }, { $inc: { counts: +1 } });
    if (result.acknowledged) {
      return '업데이트 성공';
    } else {
      throw new Error('통신 이상');
    }
  },
  setData: async () => {
    const client = await _client;
    const db = client.db('mbti').collection('data');
    const result = await db.insertOne(initState);
    if (result.acknowledged) {
      return '업데이트 성공';
    } else {
      throw new Error('통신 이상');
    }
  },
  getData: async () => {
    const client = await _client;
    const db = client.db('mbti').collection('data');
    const data = await db.find({}).toArray();
    return data;
  },
};

module.exports = mongoDB;
```



mongoRouter

설정

mongoRouter 설정



- 컨트롤러에 추가된 기능을 사용하는 주소 값을 라우터에 추가해 봅시다!



```
const express = require('express');
const router = express.Router();
const mongoDB = require('../controllers/mongoController');
```

```
router.post('/setdata', async (req, res) => {
  const msg = await mongoDB.setData();
  res.send(JSON.stringify(msg));
});
```

```
router.get('/count', async (req, res) => {
  const counts = await mongoDB.getCounts();
  res.send(counts);
});
```

```
router.post('/inccount', async (req, res) => {
  const msg = await mongoDB.incCounts();
  res.send(JSON.stringify(msg));
});
```

```
router.get('/getdata', async (req, res) => {
  const data = await mongoDB.getData();
  res.send(data);
});
```

```
module.exports = router;
```

Data-server/routes/mongo.js



React 에서 데이터 요청 및 사용!



useEffect 를 사용해서 데이터 받기!

- Start 페이지가 최초 마운트 되면 백엔드 서버로 데이터를 요청해서 받아 봅시다!



이제 필요할 때 호출해서 사용!

- useEffect HOOK 에 이제 필요할 때 호출해서 사용 합시다!

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

Src/pages/Start.js



MongoDB용

코드 작성



MongoDB 의 데이터를 받는 함수 작성

- `sqlDataFetch()` 함수와 같이, `mongoFetchData()` 함수를 만들어서 백에서 작성한 코드에서 데이터가 잘 넘어오는지 확인해 봅시다!



```
async function mongoFetchData() {  
  const resMongoCount = await fetch('http://localhost:4000/mongo/count');  
  if (resMongoCount.status === 200) {  
    const num = await resMongoCount.json();  
    // counts 가 제대로 들어오는지 확인하기  
    console.log(num);  
  } else {  
    throw new Error('통신 이상');  
  }  
  const resMongoData = await fetch('http://localhost:4000/mongo/getdata');  
  if (resMongoData.status === 200) {  
    const data = await resMongoData.json();  
    // 설문용 data 가 제대로 들어오는지 확인하기  
    console.log(data);  
  } else {  
    throw new Error('통신 이상');  
  }  
}
```

Src/pages/Start.jsx



들어온 데이터 확인!

- 전달된 데이터는 배열에 객체가 들어있는 형태로 들어오며, 0번 index 의 값에 우리가 원하는 데이터가 들어 있음을 확인!

```
▼ [{-}] ⓘ  
  ▼ 0:  
    counts: 8  
    id: 1  
    _id: "634bde723a75d299fee9edfd"  
    ▶ [[Prototype]]: Object  
    length: 1  
    ▶ [[Prototype]]: Array(0)  
  
▼ [{-}] ⓘ  
  ▼ 0:  
    ▶ explanation: {ESTJ: {-}, ISTJ: {-}, ENTJ: {-}, INTJ: {-}, ESFJ: {-}, ...}  
    mbtiResult: ""  
    page: 0  
    ▶ survey: (4) [{-}, {-}, {-}, {-}]  
    _id: "634cb653a123bd5eeafc1f35"  
    ▶ [[Prototype]]: Object  
    length: 1  
    ▶ [[Prototype]]: Array(0)
```




방문자수 값

전달



Counts 값 전달!

- Counts 는 값을 새로 받으면, Start 컴포넌트의 방문자 수 값을 Update 해줘야 하므로 useState 로 값을 선언 → State 값을 변경하는 Hook 에 원하는 값을 전달!



시작 페이지에서 데이터 받기



```
export default function Start() {
  const [counts, setCounts] = useState(0);
  const dispatch = useDispatch();

  async function mongoFetchData() {
    const resMongoCount = await fetch("http://localhost:4000/mongo/count");
    if (resMongoCount.status === 200) {
      const num = await resMongoCount.json();
      if (num[0].counts !== 0) setCounts(num[0].counts);
    } else {
      throw new Error("통신 이상");
    }
    const resMongoData = await fetch("http://localhost:4000/mongo/getdata");
    if (resMongoData.status === 200) {
      const data = await resMongoData.json();
      console.log(data);
    } else {
      throw new Error("통신 이상");
    }
  }
  useEffect(() => {
    fetchData();
  }, [counts]);
}
```

Src/pages/Start.jsx



```
return (  
  <>  
    <Header>개발자 MBTI 조사</Header>  
    <MainImg src="/images/main.jpg" alt="메인 이미지" />  
    <SubHeader>  
      개발자가 흔히 접하는 상황에 따라서 MBTI 를 알아 봅시다! 지금까지{" "  
        {"\n\n"}  
        {counts} 명이 참여해 주셨습니다!  
    </SubHeader>  
    <OrangeButton text="테스트 시작" clickEvent={() => dispatch(next())} />  
  </>  
);
```

Src/pages/Start.jsx



Redux Initial

Data 전달

개발자 MBTI 조사



개발자가 흔히 접하는 상황에 따라서 MBTI 를
알아 봅시다! 지금까지

0 명이 참여해 주셨습니다!

테스트 시작



InitData 를 redux 에 전달!

- 이제 초창기 redux 에서 세팅 했던 initData 와 동일한 Data 를 만들었으므로, 해당 data 를 redux 에 전달하여 초기 값으로 만들어 봅시다!
- Data 를 초기화 시켜주는 작업도 하나의 Action 이므로 redux 에 Action Type 설정 → Action 생성 함수 만들기 → 리듀서 작업의 순서로 진행해 봅시다!



Action 설정(Type, 생성 함수)

```
// 액션 타입에 INIT 추가  
const INIT = 'mbti/INIT';  
const CHECK = 'mbti/CHECK';  
const NEXT = 'mbti/NEXT';  
const RESET = 'mbti/RESET';
```

Src/store/modules/Show.js

```
// 액션 생성 함수 추가  
export function init(data) {  
  return {  
    type: INIT,  
    payload: data,  
  };  
}
```

Src/store/modules/Show.js

Redux 최초 로딩 시에 전달한 초기값 만들기!



- 리액트 앱이 최초 실행 될 때, reducer 에 전달할 빈 초기 값 설정을 해봅시다!

```
const initStateEmpty = {  
  mbtiResult: '',  
  page: 0,  
  survey: [],  
  explanation: {},  
};
```

Src/store/modules/Show.js



리듀서 작업하기!

- 빈 초기 값을 Reducer 의 State 에 전달하고, INIT 액션에 대한 동작을 구현해 봅시다!

```
export default function mbti(state = initStateEmpty, action) {  
  switch (action.type) {  
    case INIT:  
      return {  
        ...state,  
        survey: action.payload.survey,  
        explanation: action.payload.explanation,  
      };  
  }  
}
```

Src/store/modules/Show.js



받아온 Data
전달하기!



Data 전달하기!

- 만들어진 Data 를 dispatch 를 통해 init() 함수에 담아서 전달해 봅시다

```
import { next, init } from '../store/modules/mbti';
```



```
async function mongoFetchData() {
  const resMongoCount = await fetch("http://localhost:4000/mongo/count");
  if (resMongoCount.status === 200) {
    const num = await resMongoCount.json();
    if (num[0].counts !== 0) setCounts(num[0].counts);
  } else {
    throw new Error("통신 이상");
  }
  const resMongoData = await fetch("http://localhost:4000/mongo/getdata");
  if (resMongoData.status === 200) {
    const data = await resMongoData.json();
    dispatch(init(data[0]));
  } else {
    throw new Error("통신 이상");
  }
}
```

Src/pages/Start.js



Redux Initial Data 전달

- Redux 의 InitialData 는 바로 사용할 일이 없으며, 다음 MbtI 컴포넌트가 사용하면 되므로 State 값을 사용할 필요가 X
- 기존에 만들어 놓은 init() 액션 생성 함수에 받아온 데이터만 넣어서 dispatch 로 전달하면 끝!



```
async function mongoFetchData() {
  const resMongoCount = await fetch('http://localhost:4000/mongo/count');
  if (resMongoCount.status === 200) {
    const num = await resMongoCount.json();
    if (num[0].counts !== 0) setCounts(num[0].counts);
  } else {
    throw new Error('통신 이상');
  }
  const resMongoData = await fetch('http://localhost:4000/mongo/getdata');
  if (resMongoData.status === 200) {
    const data = await resMongoData.json();
    console.log(data);
    // 리액트의 기본 동작으로 통신 전 빈 데이터가 전달 되는 것을 예방
    if (data[0].survey.length !== 0) {
      dispatch(init(data[0]));
    }
  } else {
    throw new Error('통신 이상');
  }
}
```

Src/component/Start.js



마지막 페이지에서

Count + 1 요청



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

  const incCount = async () => {
    const resInc = await fetch("http://localhost:4000/mongo/inccount", {
      method: "POST",
    });
    if (resInc.status === 200) {
      console.log(await resInc.json());
    } else {
      throw new Error("통신 이상");
    }
  };

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

Src/pages/Show.js

개발자 MBTI 조사



개발자가 흔히 접하는 상황에 따라서 MBTI 를
알아 봅시다! 지금까지

1 명이 참여해 주셨습니다!

테스트 시작



YAY WE DID

**WORK TOGETHER, BELIVE IN
YOURSELF AND YOUR GROUP
MEMBER**

makeameme.org



수고하셨습니다!