[2022-11-31] 2반 실습 파일

학습 목표

- React의 전역 상태 관리 라이브러리인 Redux에 대해 배우고 이를 활용해 전역 상태를 관리해봅니다.
- 테스팅 기법에 대해 알아보고 Jest를 이용해 이를 구현합니다.

문제 풀이

1. 선 그리는 기능 추가하기

Redux는 상태 관리 툴 중 하나로 리액트에만 쓰이는 건 아니고 많은 js 프레임워크들과 같이 쓰입니다.

화요일 학습했던 useContext, userReducer와 매우 매우 비슷한 로직으로 작동합니다.

사용하게되었던 배경을 살짝 언급하자면, 하위 컴포넌트에서 사용하려면 props로 전달해주어야했고 그래서 상태 변경 관리할 때 따로 저장소를 두었습니다. 복잡한 상태관리할 때 수정할 방법(action)을 다 정의하고 그 액션에 대한 함수를 미리 정의해놔서 이거 해줘 action? 명령?을 보내면 state가 수정되었고 이는 중구난방으로 값이 수정되어 추적이 어려운걸 방지해주었습니다.

컴포넌트 레벨이 깊지 않고 작은 프로젝트의 경우 useContext, useReducer를 잘 사용하고 컴포넌트 depth가 깊어지고 상태 관리가 복잡해지고 전역적으로 관리할 상태가 많아지면 redux

를 찾게되는 것 같아요.

기존 Redux는 단점이 있었답니다. 설정이 복잡 ~ 다른 패키지 설치가 필요하고~ 같은 구조의 코드를 (boilerplate code) 많이 반복하는 문제도 있었고~

⇒ 그래서 저희를 위해서 어려운 부분은 숨기고 쉽게 조작할 수 있는 부분만 보여주는 abstraction!를 지향

그래서 오늘 수업은 그 추상화를 지향!하는! 아주 감사해야할 툴인! redux-toolkit을 사용해볼것입니다

(stackoverflow 찾아봤는데 압도적으로 redux-toolkit 승)

라이브러리 헷갈리실까봐 잠깐 용도 설명드리자면,

react — redux(redux-toolkit)은 별개의 라이브러리이고,

react-redux는 redux(redux-toolkit)을 연결해주는 역할을 합니다.

Redux-toolkit 활용 상태 관리 뼈대> (눈으로 사진 찰칵찰칵 찍으세요)

```
====================== mySlice.js

const mySlice = createSlice({
  ...
  initialState,
  reducer: {
    <<정의한액션타입1>>: (state, action) => {
      ... // mutate 방법도 가능
    }
  }

})

export const {<<정의한액션타입1>>, <<정의한앱선타입2>>} = mySlice.actions;
export default mySlice.reducer // 이걸 stored에서 myReducer라는 이름으로 import

====================== store.js

const store = configureStore({
  reducer: {
    myReducer
  }
```

```
})

====================== index.js

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

====================== 컴포넌트.js

const App = () => {
    const {initialState} = useSelector((state)=>state);
    const dispatch = useDispatch();
    return (
      <div>
          {initialState}
          <input onChange={()=>{dispatch(<<정의한액션타입1>>(데이터필요하면넣어주기))}}/>
      </div>
    )
}
```

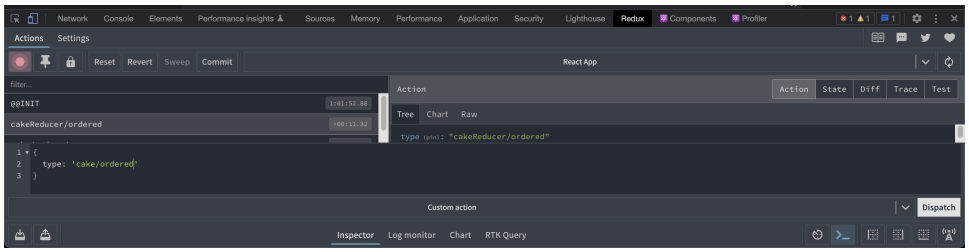
실습

> npm i @reduxjs/toolkit
> npm i react-redux
> npm i redux-logger (선택)

실습예제코드: https://codesandbox.io/s/redux-toolkit-practice-ekf4v0?file=/src/components/user/User.js

추가 + redux dev tools 설치 > 개발자 도구 redux 탭

- actions 탭: 어떠한 액션이 수행되었고 (+ payload)
- diff: 값이 어떻게 변경되었는지 나옴
- >_버튼 누르면 ui클릭 안해도 dispatch할 수 있어 (++time travel debugging도 가능)



풀이 전 접근>

- 제자리에 있고 클릭만 된 경우 점찍어주고
- 실시간으로 그려지기 보다는 첫 시작점과 끝점을 좌표값 비교해 이어주는 것 같다 이 좌표값이 상태값으로 관리될 것 같고
- 색깔 바꾸기 + 사이즈 바꾸기 기능있는 것 같고 이런걸 다 상태값으로 관리할 것 같고
- redux 상태값과 연관이 있는 걸 보니…표시된 벡터 위치에 대한 상태값이 계속 저장되어서 스크린상 보여주는 건가? 라는 생각

문제 풀이 전 힌트>

svg(벡터를 이용해 화면에 그린 그림)에서 라인을 그리려면 필요한 attribute는: x1, y1, x2, y2, stroke(선 색상), strokeWidth(선 굵기) 입니다.

handleMouseUp 마우스 뗄 때 발생하는 이벤트, handleMouseDown 마우스 클릭할 때 발생 이벤트.

▼ 문제 정답

```
// App.js

import React, { useRef } from 'react'; // useRef 역할 설명
import { configureStore, createSlice } from '@reduxjs/toolkit';
import { useDispatch, useSelector, Provider } from 'react-redux';
import styled from 'styled-components';
import { isTwoDotsSame, getPointFromEvent, getDiff, getLine } from './utils';
import { useBoundingRect } from './hooks';
```

```
const boardSlice = createSlice({
  // action(명령!)과 reducer 함수를 쉽게 생성하게 도와주는 함수 == 바로 createSlice
  name: 'board',
  initialState: {
    // lines 를 추가하세요.
    lines: [],
    dots: [],
    meta: {
      size: 4,
      color: 'black',
    },
  },
  reducers: {
    addDot: (state, action) => {
      // 함수들 통해 전달된 값이 action.payload에 담김
      state.dots.push({ // 아직 상태값이 어떻게 들어가야하는지 몰라
        ...state.meta,
        ...action.payload,
      }); // redux-toolkit 장점. immer가 내장되어있어서 불변성을 유지하기 위해서 추가적으로 코드를
 작성하지 않아도 됨
    },

    setColor: (state, action) => {
      state.meta.color = action.payload;
    },

    setSize: (state, action) => {
      state.meta.size = action.payload;
    },

    // addLine 액션을 추가하세요.
    addLines: (state, action) => {
      state.lines.push({
        ...state.meta,
        ...action.payload,
      });
    },
  },
});

const reducer = boardSlice.reducer;
const actions = boardSlice.actions;

const store = configureStore({
  reducer, //  === reducer: reducer. Es6 문법.
});

export default function App() {
  return (
    <Provider store={store}>
      <div>
        <h3>그-으림판</h3>
        <SVGBoard />
```

```
      </div>
    </Provider>
  );
}

function SVGBoard() {
  const dispatch = useDispatch(); // 함수 꺼내쓰기
  const dots = useSelector(state => state.dots); //x,y,size,color 필요하네
  const meta = useSelector(state => state.meta);
  // lines 추가
  const lines = useSelector(state => state.lines); // x1,y1,x2,y2,size,color

  // line을 useSelector를 이용해 가져오세요.
  const prevMousePositionRef = useRef(null); // **용도

  const { register, rect } = useBoundingRect(); // elRef를 지정할 함수랑 그 위치를 리턴해준듯
아직 지정은 안한 상태.. 밑에 찾아보니 DrawingPad가 렌더링 되면서 rect에 값이 들어가더라 https://seo-
tory.tistory.com/72

  const handleMouseDown = e => {
    // 마우스 누를 때
    // 이전 위치를 저장해, 위치가 다르면 선으로 인지합니다.
    prevMousePositionRef.current = getPointFromEvent(e); // 이전 좌표값을 잠깐 저장해놓는 용
도로 useRef를 쓸 수 있음
  };

  const handleMouseUp = e => {
    // 마우스 뗄 때
    const curDiff = getDiff(getPointFromEvent(e), rect); // 어떤 함수일까?
    // 마우스 떼었을 직후와 rect의 x,y좌표 기준 변화량을 반환해주는구나
    const prevDiff = getDiff(prevMousePositionRef.current, rect);
    // 이전 마우스 눌렀을 때 좌표와 rect의 x,y좌표 기준 간의 변화량

    if (isTwoDotsSame(curDiff, prevDiff)) {
      // 현재 위치, 이전 위치가 같다면 점을 찍은 거구나~ 생각
      dispatch(actions.addDot(curDiff)); // 현재 좌표 넣어주는구나
    } else {
      // 그게 다르다면 직선을 그었다고 보겠구나~
      // line을 추가합니다.
      dispatch(actions.addLines(getLine(curDiff, prevDiff))); // 두좌표를 다 넣어주는구나
      // getLine 유틸 함수를 이용하세요.
    }
  };

  const handleColorChange = e => dispatch(actions.setColor(e.target.value));
  const handleSizeChange = e =>
    dispatch(actions.setSize(Number(e.target.value)));

  return (
    <Container>
      <DrawingPad
        id="drawing_pad"
        xmlns="http://www.w3.org/2000/svg"
        {...register()}
```

```
        onMouseUp={handleMouseUp} // 눌었던 마우스를 떼어낸 이벤트
        onMouseDown={handleMouseDown} // 마우스를 눌었을 때 이벤트
      >
        {React.Children.toArray(
          // 리턴된 자식컴포넌를 key prop이 없어도 같은 depth의 배열로 변환하도록
          dots.map( // dots 원소 안에 x,y,size,color다 필요해
            // 그런데 이전에 dispatch로 curFidd 현재 좌표 넣어줬으니 size, color는 meta 데이터에서 꺼내서 넣어줘야겠다
            (
              dot // 중앙 x,y좌표 + 반지름
            ) => <circle cx={dot.x} cy={dot.y} r={dot.size} fill={dot.color} />
          )
        )}

        {/*
            예제 코드 입니다. line 데이터를 이용해 선을 그리세요.
          */}
        {React.Children.toArray( // svg에서 라인을 그리려면 필요한 attribute
          lines.map(line => (
            <line
              x1={line.x1} // 시작점
              y1={line.y1}
              x2={line.x2} // 끝나는점
              y2={line.y2}
              stroke={line.color} // 색깔
              strokeWidth={line.size} //두께
            />
          ))
        )}
      </DrawingPad>

      <Controller>
        <label htmlFor="color">
          <LabelText>Color</LabelText>
          <select
            id="color"
            name="color"
            onChange={handleColorChange}
            value={meta.color}
          >
            <option id="black" value="black">
              Black
            </option>
            <option id="red" value="red">
              Red
            </option>
            <option id="blue" value="blue">
              Blue
            </option>
            <option id="green" value="green">
              Green
            </option>
          </select>
        </label>
```

```
        <label htmlFor="size">
          <LabelText>Size</LabelText>
          <select
            id="size"
            name="size"
            onChange={handleSizeChange}
            value={meta.size}
          >
            <option id="4" value={4}>
              4
            </option>
            <option id="8" value={8}>
              8
            </option>
            <option id="12" value={12}>
              12
            </option>
            <option id="16" value={16}>
              16
            </option>
          </select>
        </label>
      </Controller>
    </Container>
  );
}

const Container = styled.div`
  background: #e9ecef;
  width: 500px;
  height: 500px;
  padding: 12px;
  display: flex;
  flex-direction: column;
`;

const DrawingPad = styled.svg`
  background: #fff9db;
  flex: 1;
`;

const Controller = styled.div`
  height: 50px;
  display: flex;
  align-items: center;

  > * + * {
    margin-left: 12px;
  }
`;

const LabelText = styled.span`
  font-size: 12px;
```

```
  margin-right: 8px;
`;
```

▼ 문제 확장

**useRef 용도

1. 특정 dom 선택 및 접근
2. 이전 값 일시 저장 용도

```
// 특정 원소 dom에서 참조시

import "./styles.css";
import { useRef, useEffect, useState } from "react";

export const App = () => {

  const refInput = useRef('');

  useEffect(()=>{
    console.log(refInput); // ref attribute를 넣어준 인풋이 참조됨을 확인 가능
    refInput.current.focus() // 포커스됨
    refInput.current.value = "메롱" // querySelector 기능으로 활용 ㅇㅇ
    // 그런데 state값이 업데이트 되는 건 아님 명!심!
    // 그리고 appendChild 가능을 통해서 돔에 또 추가할 수는 있는데 그렇게 하면 코드가 너무 복잡하고 쓰
기 어려워짐 권고 ㄴㄴ
  })
  return (
    <input ref={refInput}/>
  )
}

export default App;


// 이전 state값이 계속 업데이트 될 때 state value값을 저장하는 용도로도 사용 가능

import "./styles.css";
import React, { useRef, useEffect, useState } from "react";

export const App = () => {
  const [weight, setWeight] = useState(100);
  const preWeight = useRef(weight);

  useEffect(() => {
    preWeight.current = weight;
  }, [weight]);
```

```
  return (
    <>
      <div>이전 중량: {preWeight.current}</div>
      <button onClick={() => setWeight((weight) => weight + 10)}>증가</button>
      <div>지금 중량: {weight}</div>
      <div>감량 두둥: {preWeight.current - weight}</div>
    </>
  );
};

export default App;
```

---

2. 모두 지우기, 배경색 바꾸기 기능 추가하기

▼ 문제 정답

```
// App.js

import React, { useRef } from 'react';
import { configureStore, createSlice } from '@reduxjs/toolkit';
import { useDispatch, useSelector, Provider } from 'react-redux';
import styled from 'styled-components';
import { getPointFromEvent, isTwoDotsSame, getLine, getDiff } from './utils';
import { useBoundingRect } from './hooks';

// 글로벌 변수로 빼주기
const initialState = {
  dots: [],
  lines: [],
  meta: {
    size: 4,
    color: 'black',
    // background를 추가하세요.
    background: '#fff', // **white로 하면 에러
  },
};

const boardSlice = createSlice({
  name: 'board',
  initialState,
```

```
  reducers: {
    addDot: (state, action) => { // 설명
      state.dots.push({ ...state.meta, ...action.payload });
    },

    addLine: (state, action) => {
      state.lines.push({ ...state.meta, ...action.payload });
    },

    setColor: (state, action) => {
      state.meta.color = action.payload;
    },

    setSize: (state, action) => {
      state.meta.size = action.payload;
    },

    // resetBoard 액션을 추가하세요.
    resetBoard: (state, action) => {
      return initialState;
    },
    // setBackground 추가
    setBackground: (state, action) => {
      state.meta.background = action.payload;
    },
  },
});

const reducer = boardSlice.reducer;
const actions = boardSlice.actions;

const store = configureStore({
  reducer,
});

export default function App() {
  return (
    <Provider store={store}>
      <div>
        <h3>그-으림판</h3>
        <SVGBoard />
      </div>
    </Provider>
  );
}

function SVGBoard() {
  const dispatch = useDispatch();
  const dots = useSelector(state => state.dots);
  const lines = useSelector(state => state.lines);
  const meta = useSelector(state => state.meta);
  const prevMousePositionRef = useRef(null);

  const { register, rect } = useBoundingRect();
```

```
const handleMouseDown = e => {
  prevMousePositionRef.current = getPointFromEvent(e);
};

const handleMouseUp = e => {
  const curDiff = getDiff(getPointFromEvent(e), rect);
  const prevDiff = getDiff(prevMousePositionRef.current, rect);

  if (isTwoDotsSame(curDiff, prevDiff)) {
    return dispatch(actions.addDot(curDiff));
  }

  dispatch(actions.addLine(getLine(curDiff, prevDiff)));
};

const handleColorChange = e => dispatch(actions.setColor(e.target.value));
const handleSizeChange = e =>
  dispatch(actions.setSize(Number(e.target.value)));

const handleBackgroundChange = e =>
  dispatch(actions.setBackground(e.target.value));

return (
  <Container>
    <DrawingPad
      id="drawing_pad"
      xmlns="http://www.w3.org/2000/svg"
      {...register()}
      onMouseDown={handleMouseDown}
      onMouseUp={handleMouseUp}
      style={{
        // meta.background를 적용하세요.
        background: meta.background,
      }}
    >
      {React.Children.toArray(
        dots.map(dot => (
          <circle cx={dot.x} cy={dot.y} r={dot.size} fill={dot.color} />
        ))
      )}

      {React.Children.toArray(
        lines.map(line => (
          <line
            x1={line.x1}
            y1={line.y1}
            x2={line.x2}
            y2={line.y2}
            stroke={line.color}
            strokeWidth={line.size}
          />
        ))
      )}
```

```
      </DrawingPad>

      <Controller>
        <label htmlFor="color">
          <LabelText>Color</LabelText>
          <select // **select에서 선택된 값 꺼내오는 방법 설명. 선택이 되면 onChange
            id="color" // 선택된 값은 select 태그의 value에 저장됨
            name="color"
            onChange={handleColorChange}
            value={meta.color}
          >
            <option id="black" value="black">
              Black
            </option>
            <option id="red" value="red">
              Red
            </option>
            <option id="blue" value="blue">
              Blue
            </option>
            <option id="green" value="green">
              Green
            </option>
          </select>
        </label>

        <label htmlFor="size">
          <LabelText>Size</LabelText>
          <select
            id="size"
            name="size"
            onChange={handleSizeChange}
            value={meta.size}
          >
            <option id="4" value={4}>
              4
            </option>
            <option id="8" value={8}>
              8
            </option>
            <option id="12" value={12}>
              12
            </option>
            <option id="16" value={16}>
              16
            </option>
          </select>
        </label>

        <label htmlFor="background">
          <LabelText>Background</LabelText>
          <select
            id="background"
            name="background"
```

```
            onChange={handleBackgroundChange}
            value={meta.background}
          >
            <option id="back-white" value="#fff">
              White
            </option>
            <option id="back-green" value="#d3f9d8">
              Green
            </option>
            <option id="back-yellow" value="#fff3bf">
              Yellow
            </option>
            <option id="back-pink" value="#ffdeeb">
              Pink
            </option>
          </select>
        </label>

        <button id="reset" onClick={() => dispatch(actions.resetBoard())}>
          초기화
        </button>
      </Controller>
    </Container>
  );
}

const Container = styled.div`
  background: #e9ecef;
  width: 500px;
  height: 500px;
  padding: 12px;
  display: flex;
  flex-direction: column;
`;

const DrawingPad = styled.svg`
  background: #fff9db;
  flex: 1;
`;

const Controller = styled.div`
  height: 50px;
  display: flex;
  align-items: center;

  > * + * {
    margin-left: 12px;
  }
`;

const LabelText = styled.span`
  font-size: 12px;
  margin-right: 8px;
`;
```

3. “React로 싱글 페이지 웹 서비스 만들기” 실습의 테스트코드 만들기

테스트 코드 왜 쓰나요?

우리 의도대로 코드가 작동하는지 확인하기 위함입니다. 메뉴얼 테스팅을 하면 기능이 새로 추가되면 기존 기능들도 다시 테스트해야하고 수작업으로 다 점검할 수 없어 놓치는 부분도 있겠죠?!

왜 jest로 실습하나요?

여러 자바스크립트 테스팅 툴이 있지만 그 중 jest로 실습할 것입니다. 우선 test runner(실행) + test matcher(기대값과 같은지 비교) + test mocking(가짜?시뮬레이션?) 프레임워크까지 다 지원해줘서 편리해용

그리고 CRA시 별도의 설치없이 사용 가능합니다.

테스팅하려는 파일은 __.test.js로 작성 혹은 __test__디렉토리에 파일 위치하면 자동으로 인식해서 npm run test 커맨드 실행 시 실행됩니다.

어마무시하게 많은 matcher가 있어요 eg. 버튼이 disabled 되었는지…특정 attribute가 있는지..클래스가 있는지 등등 (공식문서 참고 요: https://github.com/testing-library/jest-dom)

▼ (참고용) jest matcher: toBe, toBeCloseTo, toEqual, mocking

toBe(단순값비교), toBeCloseTo

```
// fn.test.js

const fn = require("./fn");

test("0.1더하기 0.2는 0.3인가요", () => {
  expect(fn.add(0.1, 0.2)).toBeCloseTo(0.3); // 부동소수점! 컴퓨터는 2진수로 숫자를 저장해서 딱 떨어지는 값이 아니라 0.30000000..4가로 저장돼
// 그럴 때는 toBeCloseTo로 근사치 확인 사용할 수 있어
});
```

toEqual(js 객체 비교)

```
// fn.js
const fn = {
  makeUser: (name, age) => ({ name, age }) // 객체를 리턴하는 함수
};

module.exports = fn;

// fn.test.js

const fn = require("./fn");

test("이름과 나이를 전달 받아 객체를 반환해줘", () => {
  expect(fn.makeUser("송강", 30)).toEqual({
    name: "송강",
    age: 30
  });
});
```

mocking은…

예를 들어 유저 가입기능 테스트를 진행 할 때 테스트 할 때마다 테스트 때마다 유저가 새로 생성되지 않도록 mocking 함수 구현이라던지

시간에 따라 변할 수 있는 함수나 값은 mock함수로 대체해줄 수 있음

```
// fn.js

const fn = {
  add: (num1, num2) => num1 + num2,
  createUser: (name) => {
    console.log("사용자가 생성되었습니다.");
    return {
      name: name
    };
  }
};

module.exports = fn;

// fn.test.js

const fn = require("./fn");

jest.mock("./fn"); // 모든 함수 mocking
fn.createUser.mockReturnValue({ name: "송강" });
```

```
test("유저를 만든다", () => {
  const user = fn.createUser("송강"); // 실제 함수가 동작하지 않고 위 mocking한 함수가 동작
  expect(user.name).toBe("송강");
});
```

▼ 문제 정답

```
import { render, screen } from '@testing-library/react';
// render: 특정 컴포넌트를 받아 테스팅할 돔 생성해줌
// screen: 특정 쿼리를 활용해서 요소에 접근할 수 있음

import userEvent from '@testing-library/user-event';
import '@testing-library/jest-dom';
import App from './App';

// describe: 테스트 그룹 묶어주는 역할
describe('App.js 의 각 페이지 렌더링이 잘 이루어지는지를 확인합니다.', () => {
  test("Home 글씨를 클릭하면 '기본 페이지입니다' 문구가 적힌 컴포넌트가 화면에 나타난다.", async () =
> { // test()는 테스트명 + 함수 + timeout을 인자로 받아
    render(<App />);
    // expect(el).toMatchSnapshot(); 스냅샷 찍어 전 후 일치 여부 확인 가능

    // screen.debug(): 코드 동작 당시 dom 프린트 가능

    const user = userEvent.setup(); // 여러개 이벤트 시뮬레이션

    // userEvent.type(<<element>>, 넣을 값) <- 진짜 타이핑 한 것 처럼
    // userEvent.click(<<element>>) <- 진짜 클릭
    // testing-library에 있는 fireEvent보다 다양한 이벤트 제공

    const navbarHome = await screen.findByText('Home'); // 요소 접근 정규표현식
    await user.click(navbarHome);

    const basicPageComponent = await screen.findByText(/기본 페이지입니다/);

    expect(basicPageComponent).toBeInTheDocument(); // document에 있는지 점검
  });

  // 위 테스트 코드를 참고하여, Coffee 관련 테스트 코드를 작성해 보세요.
  test("커피 글씨를 클릭하면 '커피 관련 페이지입니다' 문구가 적힌 컴포넌트가 화면에 나타난다.", async
 () => {
    render(<App />);

    const user = userEvent.setup();

    const navbarCoffee = await screen.findByText('커피');
    await user.click(navbarCoffee);
```

```
    const coffeePageComponent = await screen.findByText(
      /커피 관련 페이지입니다/
    );

    expect(coffeePageComponent).toBeInTheDocument();
  });

  // 위 테스트 코드를 참고하여, Car 관련 테스트 코드를 작성해 보세요.
  test("자동차 글씨를 클릭하면 '커피 관련 페이지입니다' 문구가 적힌 컴포넌트가 화면에 나타난다.", async
() => {
    render(<App />);
    const user = userEvent.setup();
    const navbarCar = await screen.findByText('자동차');
    await user.click(navbarCar);

    const carPageComponent = await screen.findByText(
      /자동차 관련 페이지입니다/
    );

    expect(carPageComponent).toBeInTheDocument();
  });

  // 위 테스트 코드를 참고하여, Youtube 관련 테스트 코드를 작성해 보세요.
  test("유튜브 글씨를 클릭하면 '유튜브 관련 페이지입니다' 문구가 적힌 컴포넌트가 화면에 나타난다.", asyn
c () => {
    render(<App />);
    // screen.debug() 이 당시 렌더된 컴포넌트 보여줘
    // const app = render(<Timer/>);
    // expect(app).toMatchSnapshot();
    const user = userEvent.setup();
    const navbarYoutube = await screen.findByText('유튜브');
    await user.click(navbarYoutube);

    const youtubePageComponent = await screen.findByText(
      /유튜브 관련 페이지입니다/
    );

    expect(youtubePageComponent).toBeInTheDocument();
  });
});
```

▼ 문제 확장 (실습)

screen으로 요소 찾는 방법 (getBy-, findBy-, queryBy-로 시작하는 쿼리 어마무시

e.g. 의도적으로 데이터 가져오는데 시간이 걸려서 0.5초 뒤에 돔에 나타나도록 디자인한 경우: findBy-사용 후 timeout 설정 가능

공식 문서: https://testing-library.com/docs/

```
import {render, screen} from "@testing-library/react";
import <<컴포넌트>> from ...

test("<<test명>>", ()=>{
  render(<<컴포넌트>>);
  const txt = screen.getByText("") // 1. text 가져오기 (정규표현식 사용 가능)

  const btn = screen.getByRole("button"); // 2. 하나의 html 요소 가져오기
  // getAllBy로 일치하는 태그 모두 가져올 수도, 조건을 부여할 수도 있음

  const ipt = screen.getByDisplayValue("안뇽") // 3. value로 가져오기

  const box = screen.getByTestId("고구마") // 4. 그런데 얘는 프로젝트와 관련 없는 테스팅만을 위한
코드라서 마지막으로 사용해야함

  expect(txt).toBeInTheDocument(); // 존재하는지 확인
  expect(btn).toBeInTheDocument();
})
```

```
const Welcome = () => {
  return (
  <div>
      <div data-testid="고구마"/> // 4. 위 최후의 수단
  </div>
  )
}
```