**TodoList Project**

**주요기능**

1)전체출력

2)추가(등록)

3)수정(checkbox 상태변경)

4)삭제

5)검색

**전체 UI**

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

**Component 구조**

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

**directory구조**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

**props 전달과정**

텍스트, 도표, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

**알아야 하는 개념**

1. Component 분리
2. CSS적용
3. 이벤트처리
4. Props
5. useState
6. useRef
7. filter함수 , map함수

**React의 상태관리 Hook 적용하기**

**1)TodoList Project에 useReducer 적용하기**

useReducer는 React에서 상태 관리를 위한 Hook 중 하나로, 상태 업데이트 로직을 더 복잡하게 다뤄야 할 때 유용하다. 기본적으로 useState와 비슷하지만, 상태 업데이트가 복잡하거나 여러 액션을 처리해야 할 경우 더 적합하다.

useReducer vs useState

useReducer는 복잡한 상태 업데이트 로직이나 여러 종류의 상태 변화가 필요할 때 유용하다

예를 들어, 여러 액션을 처리하거나 상태 변화가 복잡한 경우 사용한다.

useState는 간단한 상태 관리에 적합하고, 상태 변화가 단일적인 경우 사용하기 쉽다

.

언제 useReducer를 사용할까?

* 상태 업데이트가 복잡하거나 다양한 액션에 따라 상태를 다뤄야 할 때.
* 여러 상태를 한 번에 관리하거나, 상태를 전역적으로 다루는 패턴이 필요할 때.
* 컴포넌트 로직이 복잡하거나, 자식 컴포넌트에 상태를 전달할 때, 상태 변화가 많은 경우.

useReducer를 사용하면, 상태 변화가 예측 가능하고, 관리가 더 쉬워지는 장점이 있다.

사용방법

|  |
| --- |
| const [state, dispatch] = useReducer(reducer, initialState);  useReducer는 두 개의 주요 인자를 받는다.  **첫번째 인자 - reducer 함수** : 상태를 업데이트하는 로직을 정의한 함수.  **두번째 인자 - 초기 상태** : 상태가 초기화될 때 사용할 값.  state : 현재 상태 값.  dispatch : 액션을 발생시키는 함수. 이 함수는 reducer에 액션을 전달하여 상태를 변경한다. |

**예시)**

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

**2)TodoList Project에 useCallback 적용하기**

**3)TodoList Project에 useContext 적용하기**

**4)TodoList Project에 useMemo 적용하기**