**7. Delete: 할 일 삭제하기**

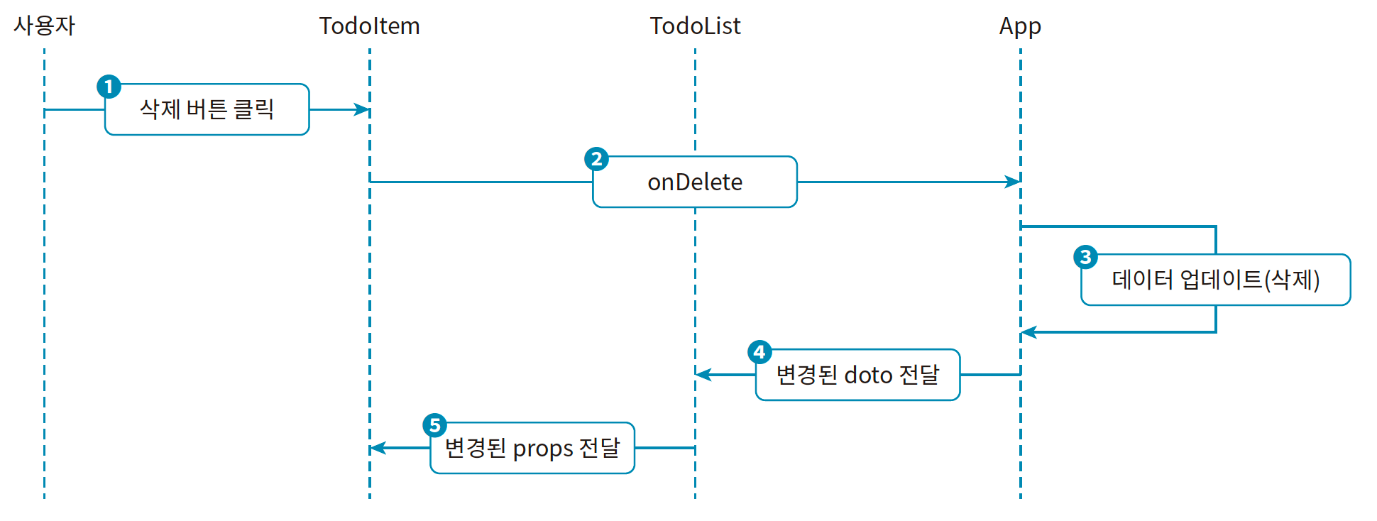
📄

**Table Of Contents**

마지막으로 CRUD의 Delete 기능을 구현하여 할 일 아이템을 삭제하겠습니다.

**기능 흐름 살펴보기**

[할 일 관리] 앱에서 삭제 기능은 어떤 흐름으로 구현 되는지 먼저 살펴보겠습니다. 할 일 아이템의 삭제는 수정 기능과 유사한 흐름으로 진행됩니다. 사용자가 TodoItem의 <삭제> 버튼을 클릭하면 해당 할 일 아이템을 찾아 삭제하면 됩니다.

[할 일 관리] 앱의 Delete 기능 흐름

1. 삭제하려는 할 일 아이템에서 <삭제> 버튼을 클릭합니다.
2. 할 일을 삭제하는 함수 onDelete를 호출합니다. 이 함수는 App의 State 값을 업데이트하므로 미리 App 컴포넌트에서 Props로 전달해야 합니다.
3. <삭제> 버튼을 클릭하면 삭제할 할 일 아이템만 빼고, 새 배열을 만들어 State 값을 업데이트합니다.
4. State 변수 todo가 업데이트되면, App가 TodoList 컴포넌트에 전달한 Props 의 값도 변경됩니다.
5. TodoList 컴포넌트는 Props의 값이 변경되면 리렌더됩니다. 이때 새로운 배 열 todo로 할 일 리스트 다시 렌더링합니다.

**아이템 삭제 함수 만들기**

App 컴포넌트에서 할 일을 삭제하는 함수 onDelete를 만듭니다.

**src/App.js**

**Copy**

(...)

function App() {

(...)

const onDelete = (targetId) => { ①

setTodo(todo.filter((it) => it.id !== targetId));

};

return (

<div className="App">

<Header />

<TodoEditor onCreate={onCreate} />

<TodoList todo={todo} onUpdate={onUpdate} onDelete={onDelete} /> ②

</div>

);

}

export default App;

**①** TodoItem의 <삭제> 버튼을 클릭 했을때 호출하는 함수 onDelete는 매개변수 targetId에 삭제 할 일기 아이템의 id를 저장합니다. 그리고 해당 id 요소를 뺀 새 배열로 todo를 업데이트 함으로써 대상 아이템을 삭제합니다.

**②** 함수 onDelete는 TodoItem에서 <삭제> 버튼을 클릭할 때 호출합니다. 따라서 먼저 TodoList에 Props로 전달해야 합니다.

TodoList는 Props로 받은 함수 onDelete를 다시 TodoItem 컴포넌트에 전달해야 합니다.

**src/component/TodoList.js**

**Copy**

(...)

const TodoList = ({ todo, onUpdate, onDelete }) => { ①

(...)

return (

<div className="TodoList">

(...)

<div className="list\_wrapper">

{getSearchResult().map((it) => (

<TodoItem

key={it.id}

{...it}

onUpdate={onUpdate}

onDelete={onDelete} ②

/>

))}

</div>

</div>

);

};

export default TodoList;

**①** Props를 구조 분해 할당합니다. 함수 onDelete를 추가합니다.

**②** 함수 onDelete를 리스트의 모든 TodoItem에 Props로 전달합니다.

**TodoItem 컴포넌트에서 삭제 함수 호출하기**

TodoItem에서 <삭제> 버튼을 클릭하면 함수 onDelete를 호출하도록 구현합니다.

**src/component/TodoItem.js**

**Copy**

import "./TodoItem.css";

const TodoItem = ({ id, content, isDone, createdDate, onUpdate, onDelete }) => { ①

const onChangeCheckbox = () => {

onUpdate(id);

};

const onClickDelete = () => { ②

onDelete(id);

};

return (

<div className="TodoItem">

<div className="checkbox\_col">

<input onChange={onChangeCheckbox} checked={isDone} type="checkbox" />

</div>

<div className="title\_col">{content}</div>

<div className="date\_col">

{new Date(createdDate).toLocaleDateString()}

</div>

<div className="btn\_col">

<button onClick={onClickDelete}>삭제</button> ③

</div>

</div>

);

};

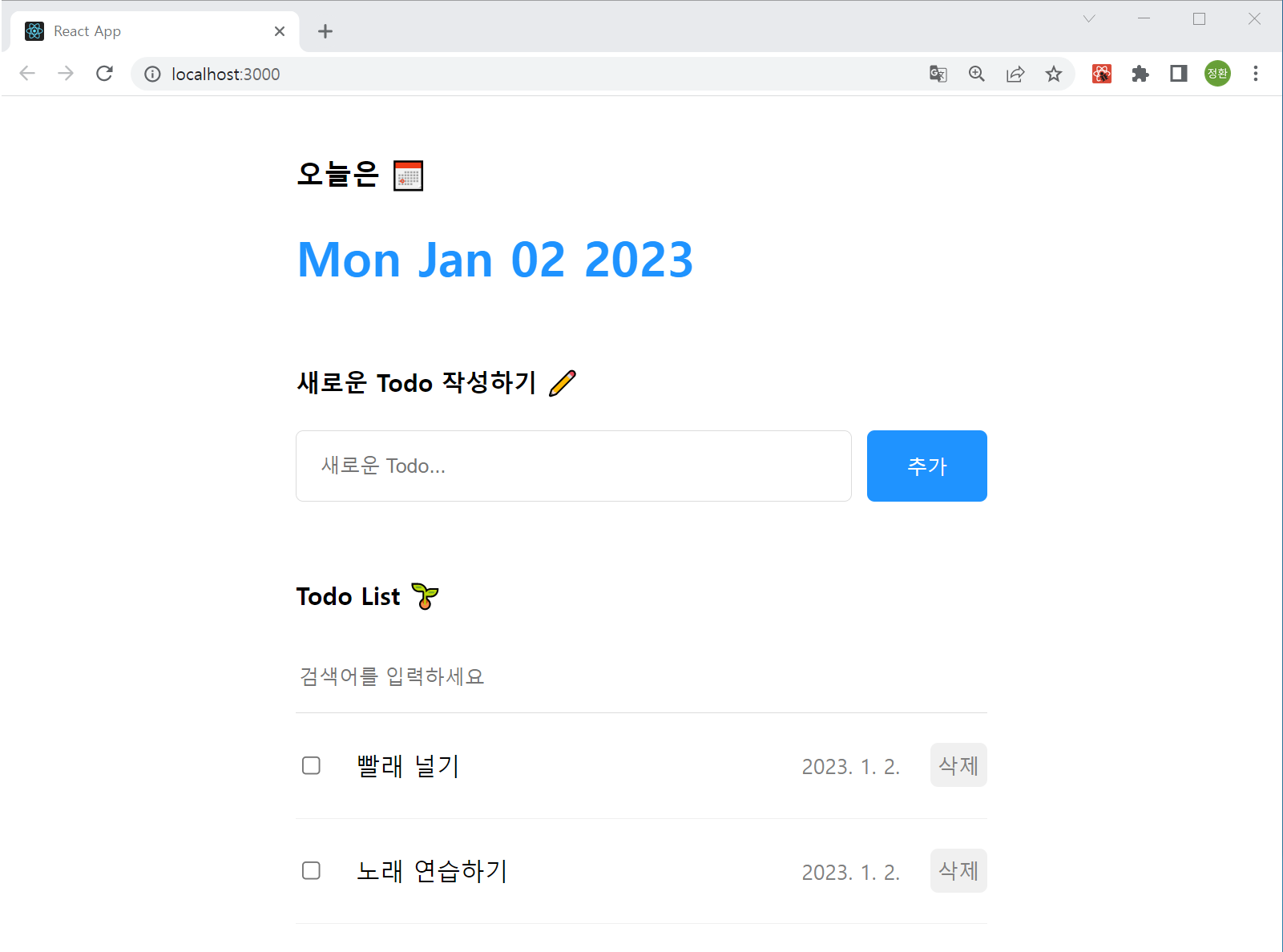
export default TodoItem;

**①** Props를 구조 분해 할당합니다. 함수 onDelete를 추가합니다.

**②** <삭제> 버튼을 클릭하면 호출할 함수 onClickDelete를 만듭니다. 이 함수는 함수 onDelete를 호출하고 인수로 해당 아이템의 id를 전달합니다.

**③** <삭제> 버튼의 onClick 이벤트 핸들러로 함수 onClickDelete를 설정합니다.

[할 일 관리] 앱에서 임의로 할 일 아이템 가운데 하나를 선택해 <삭제> 버튼을 클릭합니다. 아이템이 잘 삭제되는지 페이지에서 확인합니다.

[할 일 관리] 앱의 삭제 기능 확인하기

위 그림과 같이 <삭제> 버튼을 클릭하면 페이지에서 바로 삭제된다는 것을 알 수 있습니다.

이렇게 두 번째 리액트 앱 프로젝트인 [할 일 관리] 앱을 모두 완성했습니다. 여기까지 문제없이 완료했다면 이제는 리액트를 이용해 간단한 프로젝트 정도는 만들수 있는 기본 소양을 갖추었다고 할 수 있습니다.

그러나 Props Drilling이나 최적화 문제, 분리되지 않은 상태 관리 등 리액트 서비스와 관련해 알아야 할 내용들이 아직 더 있습니다. 다음 과정에서 이 개념들을 공부하면서 [할 일 관리] 앱을 한 단계 업그레이드하겠습니다.