recoil 기초

**학습목표**

recoil 상태 관리

recoil 비동기 조회

**목차**

[1. recoil1 프로젝트 2](#_Toc152596910)

[1) 프로젝트 생성 2](#_Toc152596911)

[2) counter 예제 - useState 3](#_Toc152596912)

[3) counter 예제 - recoil state 5](#_Toc152596913)

[4) message 예제 7](#_Toc152596914)

[2. recoil & axios 10](#_Toc152596915)

[1) callback 함수에서 발생하는 에러 10](#_Toc152596916)

[2) Error Boundary 10](#_Toc152596917)

[3) todo item 조회 - async selector 10](#_Toc152596918)

[4) todos-db.json 14](#_Toc152596919)

[5) todo item 조회 - async selectorFamily 15](#_Toc152596920)

[6) todo 목록 조회 16](#_Toc152596921)

[3. recoil로 todo 앱 구현 19](#_Toc152596922)

[1) 개요 19](#_Toc152596923)

[2) 프로젝트 생성 19](#_Toc152596924)

[3) 구현 #1 20](#_Toc152596925)

[4) 간결하게 수정 25](#_Toc152596926)

# recoil1 프로젝트

## 프로젝트 생성

|  |
| --- |
| npx create-react-app recoil1 --template typescript |

### 라이브러리 설치

|  |
| --- |
| npm i recoil |

|  |
| --- |
| npm i sass |

|  |
| --- |
| npm i axios |

### recoil

react 앱의 상태 관리를 위한 라이브러리이다.

과거에 가장 널리 사용된 대표적인 react 생태 관리 라이브러리는 redux 이다.

하지만 군더더기 코드가 많아서 구현하기 불편하다는 단점 때문에 redux의 인기가 식었다.

요즘 유행하고 있는 상태 관리 라이브러리는 facebook이 만든 recoil 이다.

깔끔하게 구현할 수 있어서 점점 인기가 많아지고 있다.

## counter 예제 - useState

### 개요

먼저 간단한 counter 앱을 useState 훅으로 구현하자.

### Counter.tsx

|  |
| --- |
| import { useState } from 'react';  function Counter() {    const [count, setCount] = useState<number>(0);    const onClick = () => setCount(count + 1);    return (      <div id="Counter">        <span>{ count }</span>        <button type="button" onClick={onClick}>++</button>      </div>    )  }  export default Counter |

### App.tsx

|  |
| --- |
| import Counter from './Counter';  import './App.scss';  function App() {    return (      <div id="App">        <Counter />        <Counter />      </div>    )  }  export default App |

### App.scss

|  |
| --- |
| #App {    width: 300px; padding: 30px; margin: 30px auto;    box-shadow: 2px 2px 6px rgba(0.0,0,0,0.4);    font-size: 11pt;    #Counter {      span { display: inline-block; padding: 10px 30px; font-size: 12pt; }      button { padding: 0.3em 1.5em }    }  } |

A screenshot of a computer

Description automatically generated with low confidence

위 화면의 두 Counter 컴포넌트 인스턴스의 count 값이 다르다.

Counter 컴포넌트 인스턴스 마다 count 상태 변수를 따로 소유하기 때문이다.

## counter 예제 - recoil state

#### index.tsx

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom/client';  import { RecoilRoot } from 'recoil'  import './index.css';  import App from './App';  import reportWebVitals from './reportWebVitals';  const root = ReactDOM.createRoot(    document.getElementById('root') as HTMLElement  );  root.render(    <React.StrictMode>      <RecoilRoot>        <App />      </RecoilRoot>    </React.StrictMode>  );  // If you want to start measuring performance in your app, pass a function  // to log results (for example: reportWebVitals(console.log))  // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals  reportWebVitals(); |

RecoilRoot는 recoil 상태를 전역에 제공해준다.

즉 RecoilRoot 아래에 여러 컴포넌트들의 recoil 상태에 쉽게 접근할 수 있다.

#### counterState.ts

|  |
| --- |
| import { atom } from 'recoil';  export const counterState = atom<number>({    key: 'counterState',    default: 0,  }); |

atom 함수: recoil 상태를 생성한다.

이 상태의 값은 number 이다.

key 속성은 이 상태의 ID 이다.

default 속성은 이 상태의 최초 값이다.

#### Counter.tsx

|  |
| --- |
| import { useRecoilState } from 'recoil';  import { counterState } from './counterState';  function Counter() {    const [count, setCount] = useRecoilState<number>(counterState);    const onClick = () => setCount(count + 1);    return (      <div id="Counter">        <span>{ count }</span>        <button type="button" onClick={onClick}>++</button>      </div>    )  }  export default Counter |

import { counterState } from './counterState';

counterState recoil 상태를 사용하기 위해 import 한다.

const [count, setCount] = useRecoilState<number>(counterState);

useRecoilState 훅은 useState 훅과 매우 비슷하다.

차이점은 다음과 같다.

- useState는 상태를 생성하지만,

useRecoilState는 상태를 생성하지 않고 recoil 전역 상태를 사용한다.

- useState는 컴포넌트 내부에 상태 변수를 만드는 것이라고 생각할 수 있다.

컴포넌트 인스턴스 마다 그 상태 변수를 따로 소유한다.

- useRecoilState는 recoil 전역 상태를 공유하며 사용한다.

counterState 전역 상태는 한 개이다. 이 상태를 사용하는 컴포넌트들은

한 개의 전역 상태를 공유한다.

#### App.tsx

|  |
| --- |
| import Counter from './Counter';  import './App.scss';  function App() {    return (      <div id="App">        <Counter />        <Counter />      </div>    )  }  export default App |

이전과 동일

#### App.scss

|  |
| --- |
| #App {    width: 300px; padding: 30px; margin: 30px auto;    box-shadow: 2px 2px 6px rgba(0.0,0,0,0.4);    font-size: 11pt;    #Counter {      span { display: inline-block; padding: 10px 30px; font-size: 12pt; }      button { padding: 0.3em 1.5em }    }  } |

이전과 동일

## message 예제

### messageState.ts

|  |
| --- |
| import { atom, selector } from 'recoil';  export const messageState = atom<string>({    key: 'messageState',    default: '안녕하세요',  });  export const messageLengthState = selector<number>({    key: 'messageLengthState',    get: ({get}) => {      const message = get(messageState);      return message.length;    },  }); |

atom<string>

string 타입의 전역 상태 변수를 한 개 생성한다.

이 상태에 값을 대입하면, 상태 변수에 값이 대입된다.

이 상태 값을 읽으면, 상태 변수의 값이 읽힌다.

selector<number>

number 타입의 계산 상태를 정의한다.

계산 상태는 상태 변수가 아니고, getter setter 함수로 구성된다.

이 상태의 값을 읽는 것은, getter 함수를 호출하여 그 리턴값을 받는 것이다.

이 상태에 값을 대입하는 것은, setter 함수를 호출하며 파라미터 값을 전달하는 것이다.

getter만 정의하면 읽을 수만 읽는 상태가 된다.

get: ({get}) => {

  const message = get(messageState);

  return message.length;

}

화살표 함수 문법으로 getter를 정의하였다.

이 화살표 함수의 파라미터는 {get} 이다. 여기서 get은 다른 상태값을 읽기 위한 get 함수이다.

const message = get(messageState);

messageState 상태값을 읽어서 message 지역 변수에 대입한다.

return message.length;

message 문자열의 길이를 리턴한다.

### MessageInput.tsx

|  |
| --- |
| import { useRecoilState } from 'recoil';  import { messageState } from './messageState'  function MessageInput() {    const [message, setMessage] = useRecoilState(messageState);    const onChange = (e: any) => setMessage(e.target.value);    return (      <div id="MessageInput">        <input type="text" value={message} onChange={onChange} />      </div>    );  }  export default MessageInput |

useState로 input 태그 입력 기능을 구현한 것과 매우 유사한 구현이다.

useState 대신 useRecoilState 훅으로 구현했으므로,

input 태그에 입력된 값은 messageState 전역 상태 변수에 대입된다.

상태값을 읽고/쓰고 할 때는 useRecoilState 훅을 사용한다.

이 훅은 [상태값, setter함수] 배열을 리턴한다.

### Message.tsx

|  |
| --- |
| import { useRecoilValue } from 'recoil';  import { messageState, messageLengthState } from './messageState'  function App() {    const message = useRecoilValue(messageState);    const messageLength = useRecoilValue(messageLengthState);    return (      <div id="Message">        <p>{ message }</p>        <p>{ messageLength }</p>      </div>    )  }  export default App |

상태값을 읽기만 할 때는 useRecoilValue 훅을 사용한다.

이 훅은 상태값만 리턴한다.

### MessageInit.tsx

|  |
| --- |
| import { useSetRecoilState } from 'recoil';  import { messageState } from './messageState';  function MessageInit() {    const setMessageState = useSetRecoilState(messageState);    const onClick = () => setMessageState("안녕하세요");    return (      <div id="MessageInit">        <button type="button" onClick={onClick}>초기화</button>      </div>    )  }  export default MessageInit |

상태값을 쓰기만 할 때는 useSetRecoilState 훅을 사용한다.

이 훅은 setter 함수만 리턴한다.

### App.tsx

|  |
| --- |
| import MessageInput from './MessageInput';  import MessageInit from './MessageInit';  import Message from './Message';  import './App.scss';  function App() {    return (      <div id="App">        <Message />        <MessageInput />        <MessageInit />      </div>    )  }  export default App |

### App.scss

|  |
| --- |
| #App {    width: 300px; padding: 30px; margin: 30px auto;    box-shadow: 2px 2px 6px rgba(0.0,0,0,0.4);    font-size: 11pt;    #MessageInput {      input { padding: 5px; font-size: 11pt; }      margin: 15px 0;    }    #MessageInit {      button { padding: 0.3em 1.5em }    }    #Message {      border: 1px solid #aaa;      padding: 0 10px;    }  } |

# recoil & axios

## callback 함수에서 발생하는 에러

try {

setTimeout(() => { ..... }, 1000);

} catch (error) {

}

..... 콜백함수 본문이고, 이 콜백함수는 1000밀리터 초 후에 호출된다.

이 콜백함수에서 발생한 에러는 catch 되지 않는다.

## Error Boundary

자식 컴포넌트들에서 발생한 에러를 catch 하는 역할을 하는 ErrorBoundary 컴포넌트를 사용하자.

|  |
| --- |
| npm install react-error-boundary |

ErrorBoundary 컴포넌트 라이브러리 설치

## todo item 조회 - async selector

### todoState.ts

|  |
| --- |
| import { atom, selector } from 'recoil';  import axios from 'axios';  axios.defaults.baseURL = "http://localhost:3000";  export const todoIdState = atom<number>({    key: 'todoIdState',    default: 1  });  export const todoItemQuery = selector({    key: 'todoItemQuery',    get: async ({ get }) => {      const id = get(todoIdState);      const response = await axios.get("/todos/" + id);      return response.data;    }  }) |

export const todoIdState = atom<number>({

  key: 'todoIdState',

  default: 1

});

todo id 값을 저장하기 위한 atom 상태. 디폴트 값은 1.

export const todoItemQuery = selector({

  key: 'todoItemQuery',

  get: async ({ get }) => {

    const id = get(todoIdState);

    const response = await axios.get("/todos/" + id);

    return response.data;

  }

})

getter 함수로 정의되는 selector 상태.

이 getter 함수는 axios 호출을 하는 async 함수이다.

todoIdState 상태값이 id인 todo 객체를 조회해서 리턴한다.

즉 id 값이 1인 todo 객체를 조회해서 리턴한다.

### TodoItem.tsx

|  |
| --- |
| import { todoItemQuery } from "./todoState";  import { useRecoilValue } from "recoil";  function TodoItem() {    const todo = useRecoilValue(todoItemQuery);    return (      <div>        <h1>TodoItem</h1>        <p>{ todo.id }</p>        <p>{ todo.title }</p>        <p>{ todo.done ? "완료" : "미완료" }</p>      </div>    )  }  export default TodoItem |

todoItemQuery 상태값을 출력한다.

즉 id 값이 1인 todo 객체를 조회해서 리턴한다.

### App.scss

|  |
| --- |
| #App {    width: 300px; padding: 30px; margin: 30px auto;    box-shadow: 2px 2px 6px rgba(0.0,0,0,0.4);    font-size: 11pt;  } |

### App.tsx

|  |
| --- |
| import { Suspense } from "react";  import { ErrorBoundary } from 'react-error-boundary';  import TodoItem from './TodoItem';  import './App.scss';  function ErrorHandler({error} : {error: any}) {    return <div>에러: { error.message ? error.message : error }</div>  }  function Loading() {    return <div>Loading...</div>  }  function App() {    return (      <div id="App">        <ErrorBoundary FallbackComponent={ErrorHandler}>          <Suspense fallback={<Loading />}>            <TodoItem />          </Suspense>        </ErrorBoundary>      </div>    )  }  export default App |

<ErrorBoundary FallbackComponent={ErrorHandler}>

자식 컴포넌트들에서 catch 되지 않은 에러가 발생했을 때,

ErrorHandler 컴포넌트가 출력된다.

이 컴포넌트에 error prop으로 error 값이 전달된다.

<Suspense fallback={<Loading />}>

자식 컴포넌트들에서 백엔드를 호출하였고 아직 응답이 도착하지 않았을 때,

<Loading /> 컴포넌트가 출력된다.

### 실험

json-server를 종료하고 실행해보자.

## todos-db.json

|  |
| --- |
| {    "todos": [      {        "id": 1,        "title": "프론트엔드 과제",        "done": false      },      {        "id": 2,        "title": "백엔드 과제",        "done": true      },      {        "id": 3,        "title": "웹프 시험공부",        "done": false      }    ]  } |

백엔드 실행

|  |
| --- |
| json-server --watch todos-db.json |

프론트엔드 실행

|  |
| --- |
| npm run start |

## todo item 조회 - async selectorFamily

### todoState.ts

|  |
| --- |
| import { selectorFamily } from 'recoil';  import axios from 'axios';  axios.defaults.baseURL = "http://localhost:3000";  export const todoItemQuery = selectorFamily({    key: 'todoItemQuery',    get: (id: number) => async () => {      const response = await axios.get("/todos/" + id);      return response.data;    }  }) |

selector 상태는 getter 함수에 파라미터가 없지만,

selectorFamily 상태는 getter 함수에 파라미터가 있다.

getter 함수에 파라미터 값을 전달해야 할 때,

selector 상태 대신, selectorFamily 상태를 사용하면 된다.

### TodoItem.tsx

|  |
| --- |
| import { todoItemQuery } from "./todoState";  import { useRecoilValue } from "recoil";  function TodoItem() {    const todo = useRecoilValue(todoItemQuery(2));    return (      <div>        <h1>TodoItem</h1>        <p>{ todo.id }</p>        <p>{ todo.title }</p>        <p>{ todo.done ? "완료" : "미완료" }</p>      </div>    )  }  export default TodoItem |

id 값이 2인 todo 객체가 조회되어 출력된다.

## todo 목록 조회

### todoState.ts

|  |
| --- |
| import { selector } from 'recoil';  import axios from 'axios';  axios.defaults.baseURL = "http://localhost:3000";  export type Todo = {    id: string,    title: string,    done: boolean  }  export const todoListQuery = selector({    key: 'todoListQuery',    get: async () => {      const response = await axios.get("/todos/");      return response.data;    }  }) |

todoListQuery 상태는 todo 전체 목록을 조회해서 리턴한다.

### TodoItem.tsx

|  |
| --- |
| import type { Todo } from './todoState';  type Props = {    todo: Todo  }  function TodoItem({ todo }: Props) {    return (      <tr className={ todo.done ? "done" : "" }>        <td>{ todo.id }</td>        <td>{ todo.title }</td>      </tr>    )  }  export default TodoItem; |

### TodoList.tsx

|  |
| --- |
| import type { Todo } from './todoState'  import TodoItem from './TodoItem'  type Props = {    todos: Todo[]  }  function TodoList({ todos }: Props) {    const trlist = todos.map(todo =>      <TodoItem key={todo.id} todo={todo} />);    return (      <table>        <thead>          <tr><td>id</td><td>할일</td></tr>        </thead>        <tbody>          {trlist}        </tbody>      </table>    )  }  export default TodoList |

### TodoRoot.tsx

|  |
| --- |
| import { todoListQuery } from "./todoState";  import { useRecoilValue } from "recoil";  import TodoList from './TodoList';  function TodoRoot() {    const todos = useRecoilValue(todoListQuery);    return (      <div id="TodoRoot">        <h1>할일</h1>        <TodoList todos={todos} />      </div>    )  }  export default TodoRoot; |

### App.tsx

|  |
| --- |
| import { Suspense } from "react";  import { ErrorBoundary } from 'react-error-boundary';  import './App.scss';  import TodoRoot from "./TodoRoot";  function ErrorHandler({error} : {error: any}) {    return <div>에러: { error.message ? error.message : error }</div>  }  function Loading() {    return <div>Loading...</div>  }  function App() {    return (      <div id="App">        <ErrorBoundary FallbackComponent={ErrorHandler}>          <Suspense fallback={<Loading />}>            <TodoRoot />          </Suspense>        </ErrorBoundary>      </div>    )  }  export default App |

### App.scss

|  |
| --- |
| #App {    border: 1px solid gray; width: 400px; margin: 10px auto;    box-shadow: 5px 5px 5px #DDD; padding: 30px;    #TodoRoot {      h1 { margin: 0  0 20px 0; }      table { border-collapse: collapse; margin: 20px 0; width: 100%; }      td { border: 1px solid gray; padding: 6px; }      thead { background-color: #eee; text-align: center; }      td:nth-child(1) { text-align: center; width: 30px; }      tr.done { color: #bbb; text-decoration: line-through; }    }  } |

A screenshot of a computer

Description automatically generated with low confidence

# recoil로 todo 앱 구현

## 개요

recoil로 구현하는 상태는 대부분,

백엔드에 저장하지는 않고 프론트엔드앱 내부에서만 관리하는 전역 상태들이다.

그래서 Todo 목록을 백엔드에 저장하지 않고,

프론트엔드앱 내부에서만 관리하는 전역 상태로 구현해 보자.

## 프로젝트 생성

|  |
| --- |
| npx create-react-app todo7 --template typescript |

|  |
| --- |
| npm i recoil |

## 구현 #1

### index.tsx

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom/client';  import { RecoilRoot } from 'recoil'  import './index.css';  import App from './App';  import reportWebVitals from './reportWebVitals';  const root = ReactDOM.createRoot(    document.getElementById('root') as HTMLElement  );  root.render(    <React.StrictMode>      <RecoilRoot>        <App />      </RecoilRoot>    </React.StrictMode>  );  // If you want to start measuring performance in your app, pass a function  // to log results (for example: reportWebVitals(console.log))  // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals  reportWebVitals(); |

### todoListState.ts

|  |
| --- |
| import { atom } from 'recoil';  export type Todo = {    id: number,    title: string,    done: boolean  }  export const todoListState = atom<Todo[]>({    key: 'todoListState',    default: [      {id: 1, title: '프론트엔드 과제', done: false},      {id: 2, title: '백엔드 과제', done: true},      {id: 3, title: '웹프 시험공부', done: false}    ]  });  export type SetTodoListFunc = (todoList: Todo[]) => void;  export function addTodo(title: string, todoList: Todo[], setTodoList: SetTodoListFunc) {    let lastId = todoList.length === 0 ? 0 : todoList[todoList.length - 1].id;    let todo: Todo = { id: lastId + 1, title: title, done: false };    let newTodoList = [...todoList, todo];    setTodoList(newTodoList);  }  export function toggleTodo(id: number, todoList: Todo[], setTodoList: SetTodoListFunc) {    let newTodoList = todoList.map(todo =>        (id !== todo.id) ? todo :  {...todo, done: !todo.done });    setTodoList(newTodoList);  }  export function deleteTodo(id: number, todoList: Todo[], setTodoList: SetTodoListFunc) {    let newTodoList = todoList.filter(todo => todo.id !== id );    setTodoList(newTodoList);  } |

### TodoItem.tsx

|  |
| --- |
| import React from 'react'  import { useRecoilState } from 'recoil';  import type { Todo } from './todoListState';  import { todoListState, toggleTodo, deleteTodo } from './todoListState';  type Props = {    todo: Todo  }  function TodoItem({ todo }: Props) {    const [todoList, setTodoList] = useRecoilState<Todo[]>(todoListState);    const onChange = () => toggleTodo(todo.id, todoList, setTodoList)    const onClick = () => {      if (window.confirm('삭제하시겠습니까?'))        deleteTodo(todo.id, todoList, setTodoList)    }    return (      <tr className={ todo.done ? "done" : "" }>        <td>{ todo.id }</td>        <td>          <input type='checkbox' checked={ todo.done } onChange={onChange} />          { todo.title }          <span onClick={onClick}>x</span>        </td>      </tr>    )  }  export default TodoItem; |

useRecoilState로 구현할 때는, props로 함수들을 전달할 필요가 없다.

컴포넌트들이 todoListState 전역 상태에 직접 접근할 수 있기 때문이다.

### TodoInput.tsx

|  |
| --- |
| import React, { ChangeEvent, useState } from 'react'  import { useRecoilState } from 'recoil';  import type { Todo } from './todoListState';  import { todoListState, addTodo } from './todoListState';  function TodoInput() {    const [todoList, setTodoList] = useRecoilState<Todo[]>(todoListState);    const [title, setTitle] = useState<string>('');    const onChange = (e: ChangeEvent<HTMLInputElement>) => setTitle(e.target.value);    const onClick = () => {      addTodo(title, todoList, setTodoList);      setTitle('');    }    return (      <div>        <input type='text' value={title} onChange={onChange} />        <button onClick={onClick}>추가</button>      </div>    )  }  export default TodoInput |

### TodoList.tsx

|  |
| --- |
| import React from 'react'  import { useRecoilState } from 'recoil';  import type { Todo } from './todoListState';  import { todoListState } from './todoListState';  import TodoItem from './TodoItem'  function TodoList() {    const [todoList] = useRecoilState<Todo[]>(todoListState);    const trlist = todoList.map(todo => <TodoItem todo={todo} key={todo.id} />);    return (      <table>        <thead>          <tr><td>id</td><td>할일</td></tr>        </thead>        <tbody>          {trlist}        </tbody>      </table>    )  }  export default TodoList |

### TodoRoot.tsx

|  |
| --- |
| import React from 'react'  import './TodoRoot.css'  import TodoInput from './TodoInput';  import TodoList from './TodoList';  function TodoRoot() {    return (      <div className="todoRoot">        <h1>할일</h1>        <TodoInput />        <TodoList />      </div>    )  }  export default TodoRoot; |

### TodoRoot.css

|  |
| --- |
| .todoRoot { border: 1px solid gray; width: 400px; margin: 10px auto;    box-shadow: 5px 5px 5px #DDD; padding: 30px; }  .todoRoot h1 { margin: 0  0 20px 0; }  .todoRoot div > input { padding: 6px; width: 280px; }  .todoRoot div button { margin-left: 5px; padding: 3px 10px; }  .todoRoot table { border-collapse: collapse; margin-top: 20px; width: 100%; }  .todoRoot td { border: 1px solid gray; padding: 6px; }  .todoRoot thead { background-color: #eee; text-align: center; }  .todoRoot td:nth-child(1) { text-align: center; width: 30px; }  .todoRoot input[type=checkbox] { margin-right: 10px; accent-color: gray; }  .todoRoot span { float: right; font-weight: bold; cursor: pointer; }  .todoRoot tr.done { color: #bbb; text-decoration: line-through; } |

### App.tsx

|  |
| --- |
| import React from 'react';  import TodoRoot from './TodoRoot'  function App() {    return (      <div className="App">        <TodoRoot />      </div>    );  }  export default App; |

## 간결하게 수정

### TodoItem.tsx

|  |
| --- |
| import React from 'react'  import { useRecoilState } from 'recoil';  import type { Todo } from './todoListState';  import { todoListState, toggleTodo, deleteTodo } from './todoListState';  type Props = {    todo: Todo  }  function TodoItem({ todo }: Props) {    const todoListValueAndSetter = useRecoilState<Todo[]>(todoListState);    const onChange = () => toggleTodo(todo.id, ...todoListValueAndSetter)    const onClick = () => {      if (window.confirm('삭제하시겠습니까?'))        deleteTodo(todo.id, ...todoListValueAndSetter)    }    return (      <tr className={ todo.done ? "done" : "" }>        <td>{ todo.id }</td>        <td>          <input type='checkbox' checked={ todo.done } onChange={onChange} />          { todo.title }          <span onClick={onClick}>x</span>        </td>      </tr>    )  }  export default TodoItem; |

### TodoInput.tsx

|  |
| --- |
| import React, { ChangeEvent, useState } from 'react'  import { useRecoilState } from 'recoil';  import type { Todo } from './todoListState';  import { todoListState, addTodo } from './todoListState';  function TodoInput() {    const todoListValueAndSetter = useRecoilState<Todo[]>(todoListState);    const [title, setTitle] = useState<string>('');    const onChange = (e: ChangeEvent<HTMLInputElement>) => setTitle(e.target.value);    const onClick = () => {      addTodo(title, ...todoListValueAndSetter);      setTitle('');    }    return (      <div>        <input type='text' value={title} onChange={onChange} />        <button onClick={onClick}>추가</button>      </div>    )  }  export default TodoInput |