



YouTube

NAVER 카페

구멍가게 코딩단

# 코드로 배우는 리액트

## 12. 리액트 쿼리와 리코일

# 12장. 리액트 쿼리와 리코일

- 비동기 처리를 보다 손쉽게 할 수 있는 리액트 쿼리(React Query)와 리덕스 대신에 많이 사용하는 상태관리 라이브러리인 리코일(Recoil)을 이용
- 리액트 쿼리의 경우 기존의 서버의 데이터를 캐싱하는 기능을 가지고 있어서 불필요한 서버 호출을 줄여 줄 뿐 아니라 간략한 코드로 기능을 구성
- 리코일은 과거 리덕스(리덕스 툴킷 이전)의 복잡함을 없애고 손쉽게 애플리케이션의 상태 관리를 가능

## 개발목표

1. 리액트 쿼리(React Query)의 설정과 상품 관리 적용
2. 리코일(Recoil)의 설정과 로그인 처리 적용

# Index

12.1 리액트 쿼리

12.2 상품 목록 페이지

12.3 상품 등록 처리

12.4 상품 수정 처리

12.5 리코일(Recoil) 라이브러리

12.6 장바구니 처리

12.7 로그아웃 처리

## 리액트 쿼리의 설정

→ 리액트 쿼리 라이브러리 설치

```
npm i @tanstack/react-query  
npm i @tanstack/react-query-devtools
```


◆ .gitignore  
{ } package-lock.json  
{ } package.json

```
"dependencies": {  
  "@reduxjs/toolkit": "^1.9.6",  
  "@tanstack/react-query": "^4.35.7",  
  "@tanstack/react-query-devtools": "^4.35.7",  
  "@testing-library/jest-dom": "^5.17.0",  
  "@testing-library/react": "^13.4.0",  
  "@testing-library/user-event": "^13.5.0",  
}
```

## 리액트 쿼리의 설정

→ QueryClient를 지정 : App.js

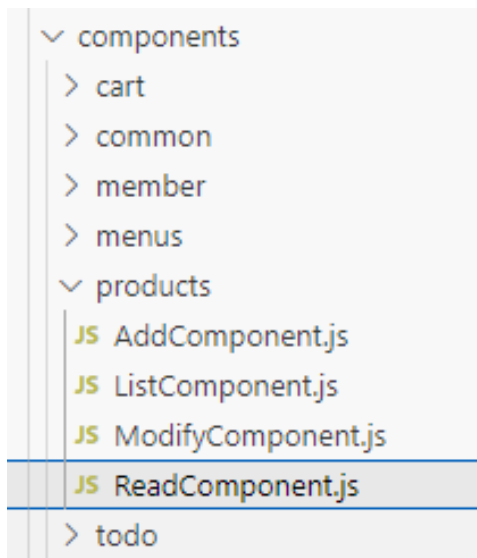
```
> util  
# App.css  
JS App.js  
JS App.test.js
```



```
import {RouterProvider} from "react-router-dom";  
import root from "../router/root";  
import { QueryClient, QueryClientProvider } from "@tanstack/react-query";  
import { ReactQueryDevtools } from "@tanstack/react-query-devtools";  
  
const queryClient = new QueryClient()  
  
function App() {  
  return (  
    <QueryClientProvider client={queryClient}>  
  
      <RouterProvider router={root}/>  
  
      <ReactQueryDevtools initialIsOpen={true} />  
  
    </QueryClientProvider>  
  );  
}  
export default App;
```

## 리액트 쿼리의 설정

→ useQuery를 이용한 상품 조회



```
import {getOne} from "../../api/productsApi"
import { API_SERVER_HOST } from "../../api/todoApi"
import useCustomMove from "../../hooks/useCustomMove"
import FetchingModal from "../../common/FetchingModal"
import { useQuery } from "@tanstack/react-query"

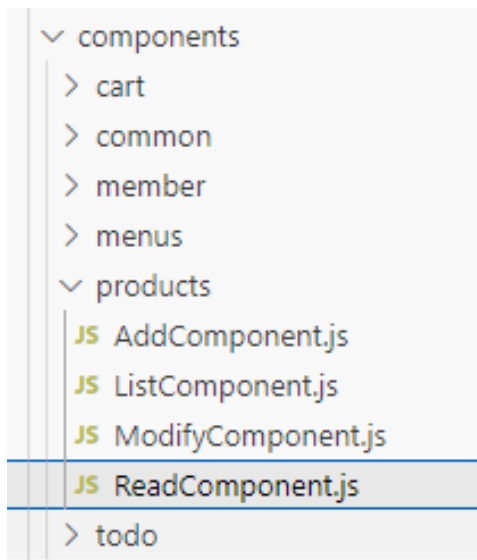
const initState = {pno:0, pname:'', pdesc:'', price: 0, uploadFileNames:[] }
const host = API_SERVER_HOST

const ReadComponent = ({pno }) => {
  const {moveToList, moveToModify} = useCustomMove()
  const {isFetching, data} = useQuery(
    ['products', pno],
    () => getOne(pno),
    {
      staleTime: 1000 * 10,
      retry: 1
    }
  )

  const handleClickAddCart = () => {
  }
```

## 리액트 쿼리의 설정

→ useQuery를 이용한 상품 조회



```
const product = data || initState
return (
  <div className = "border-2 border-sky-200 mt-10 m-2 p-4">
    {isFetching? <FetchingModal/> :<></>} ←-----
    <div className="flex justify-center mt-10">
      ...생략
    </div>
  )
}
export default ReadComponent
```

## 리액트 쿼리의 설정

→ staleTime과 쿼리 키(key)

- 리액트 쿼리의 핵심 개념:

데이터를 보관하며 상태에 따라 데이터를 가져오거나 보관된 데이터를 활용함.

- 데이터 상태 변화:

리액트 쿼리 개발 툴에서 'fetching' 상태는 데이터를 가져오는 동안, 'fresh' 상태는 데이터 처리가 완료된 후를 나타냄.

- staleTime의 활용:

코드 예제에서 'staleTime'을 통해 데이터의 신선도를 지정함.

'staleTime'이 지나면 데이터를 다시 가져옴.

- 상태 변화 및 서버 호출:

개발자 도구를 통해 'fetching -> fresh -> 10초 후 stale' 상태를 확인할 수 있음.

리액트 쿼리는 현재 화면이 활성화될 때 다시 서버를 호출하며, 다른 프로그램 선택 후에는 일정 시간 동안 서버를 호출하지 않음.



## 리액트 쿼리의 설정

→ staleTime과 쿼리 키(key)

- **useQuery의 자동 서버 호출:**

'useQuery'는 기본적으로 브라우저가 활성화될 때 서버를 다시 호출하는 옵션이 있음.

'stale' 상태의 데이터는 오래된 것으로 간주되어 다시 호출됨.

- **staleTime 설정에 따른 동작:**

'staleTime'이 짧으면 상태가 빨리 'stale'로 변경되어 서버를 다시 호출함.

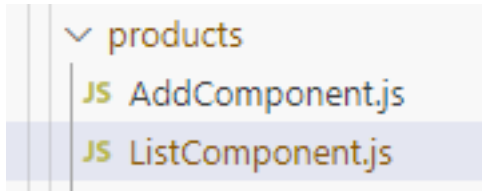
'staleTime'이 길면 상태가 오랫동안 유지되어 서버를 다시 호출하지 않음.

- **추가 참고 사항:**

'useQuery'의 다양한 옵션은 <https://tanstack.com/query/v4/docs/react/reference/useQuery>에서 확인 가능함.

## 중복적인 쿼리 키(key)

→ components/products/ListComponent.js 수정

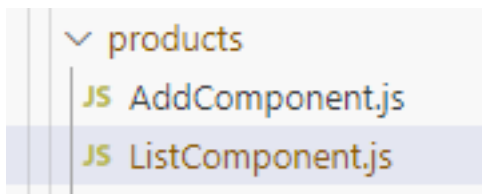


```
import { getList } from "../../api/productsApi";
import useCustomMove from "../../hooks/useCustomMove";
import FetchingModal from "../../common/FetchingModal";
import { API_SERVER_HOST } from "../../api/todoApi";
import PageComponent from "../../common/PageComponent";
import useCustomLogin from "../../hooks/useCustomLogin";
import { useQuery } from "@tanstack/react-query";

const initState = {
  dtoList:[], pageNumList:[],
  pageRequestDTO: null,
  prev: false, next: false,
  totoalCount: 0,
  prevPage: 0,
  nextPage: 0,
  totalPage: 0,
  current: 0
}
```

## 중복적인 쿼리 키(key)

→ components/products/ListComponent.js 수정



```
const host = API_SERVER_HOST

const ListComponent = () => {
  const {moveToLoginReturn} = useCustomLogin()
  const {page, size, refresh, moveToList, moveToRead} = useCustomMove()

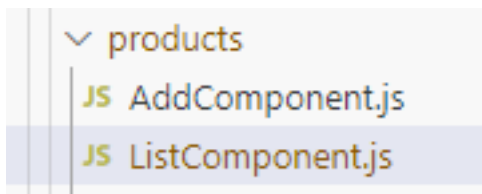
  const {isFetching, data, error, isError} = useQuery(
    ['products/list' , {page,size}],
    () => getList({page,size})
  )

  if(isError) {
    console.log(error)
    return moveToLoginReturn()
  }

  const serverData = data || initState
```

## 중복적인 쿼리 키(key)






→ components/products/ListComponent.js 수정







```
return (  
  <div className="border-2 border-blue-100 mt-10 mr-2 ml-2">  
    {isFetching? <FetchingModal/> : <></>}  
    <div className="flex flex-wrap mx-auto p-6">  
      {serverData.dtoList.map(product =>  
        ...생략  
      )}  
    </div>  
  
    <PageComponent serverData={serverData} movePage={moveToList}>  
    </PageComponent>  
  
  </div>  
  
  );  
}  
  
export default ListComponent;
```

## 중복적인 쿼리 키(key)

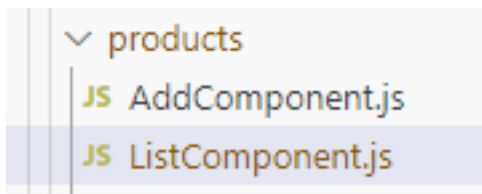
→ 동일 페이지 갱신 문제

<input type="checkbox"/> items	200
<input type="checkbox"/> list?page=2&size=10	200
 s_05b3475b-c124-4115-a1e6-a15291703839_IMG_0156.jpg	200
 s_98923128-ae48-48a2-9165-6088ac0fa847_M14.jpeg	200
 s_333d7f04-8003-4169-b4f1-3ad214dec346_M27.jpeg	200
 s_48b1bad1-6dc5-4616-9ec5-f1bc59f7040d_IMG_0161.jpg	200
 s_f8f4cb43-4037-45f9-b467-5fd9ff6f592a_M24.jpeg	200
<input type="checkbox"/> ws	101
<input type="checkbox"/> list?page=2&size=10	200
<input type="checkbox"/> list?page=2&size=10	200

 s_98923128-ae48-48a2-9165-6088ac0fa847_M14.jpeg	200	jpeg
 s_333d7f04-8003-4169-b4f1-3ad214dec346_M27.jpeg	200	jpeg
 s_48b1bad1-6dc5-4616-9ec5-f1bc59f7040d_IMG_0161.jpg	200	jpeg
 s_f8f4cb43-4037-45f9-b467-5fd9ff6f592a_M24.jpeg	200	jpeg
<input type="checkbox"/> list?page=2&size=10	200	xhr

## 중복적인 쿼리 키(key)

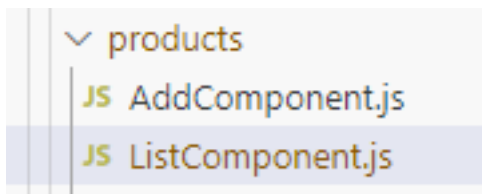
→ invalidateQueries( )



```
import { useQuery, useQueryClient } from "@tanstack/react-query";
...생략
const {isFetching, data, error, isError} = useQuery(
  ['products/list' , {page,size}],
  () => getList({page,size})
)
const queryClient = useQueryClient() //리액트 쿼리 초기화를 위한 현재 객체
const handleClickPage = (pageParam) => {
  if(pageParam.page === parseInt(page)){
    queryClient.invalidateQueries("products/list")
  }
  moveToList(pageParam)
}
..생략
//movePage 속성값으로는 handleClickPage를 전달
<PageComponent serverData={serverData} movePage={handleClickPage}>
</PageComponent>
```

## 중복적인 쿼리 키(key)

→ refresh 활용



```
const {isFetching, data, error, isError} = useQuery(
  ['products/list' , {page,size, refresh}], //←-----refresh 추가
  () => getList({page,size}),
  {staleTime: 1000 * 5 } // ←----- staleTime 추가
)

//const queryClient = useQueryClient() //필요하지 않음

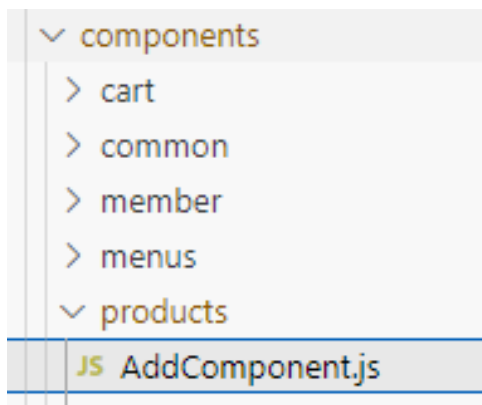
const handleClickPage = (pageParam) => {

  // if(pageParam.page === parseInt(page)){
  //   queryClient.invalidateQueries("products/list")
  // }

  moveToList(pageParam)
}
```

## 장바구니 아이템 컴포넌트

→ components/products/AddComponent.js 수정



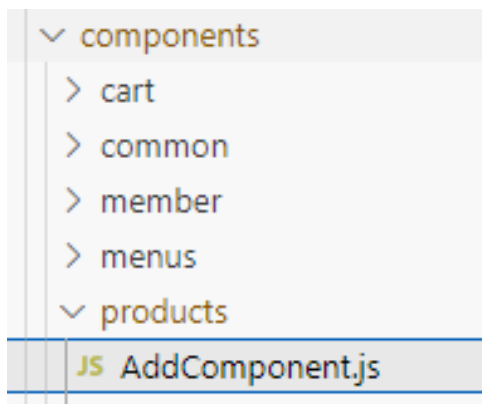
```
import { useRef, useState } from "react";
import { postAdd } from "../../api/productsApi";
import FetchingModal from "../../common/FetchingModal";
import ResultModal from "../../common/ResultModal";
import useCustomMove from "../../hooks/useCustomMove";
import { useMutation } from "@tanstack/react-query";

const initState = {
  pname: '',
  pdesc: '',
  price: 0,
  files: []
}
```



## 장바구니 아이템 컴포넌트

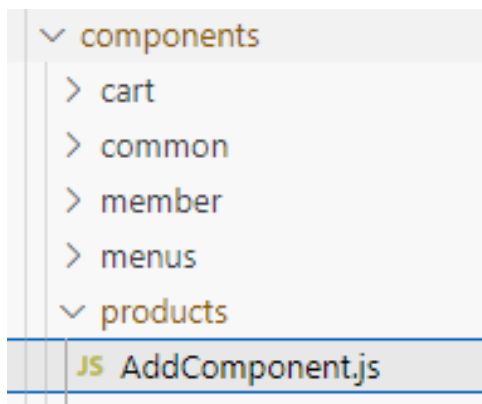
→ components/products/AddComponent.js 수정



```
const AddComponent = () => {  
  //기본적으로 필요  
  const [product, setProduct] = useState({...initState})  
  const uploadRef = useRef()  
  const {moveToList} = useCustomMove()  
  
  //입력값 처리  
  const handleChangeProduct = (e) => {  
    product[e.target.name] = e.target.value  
    setProduct({...product})  
  }  
  
  //리액트 쿼리  
  const addMutation = useMutation( (product) => postAdd(product))
```

## 장바구니 아이템 컴포넌트

→ components/products/AddComponent.js 수정



```
const handleClickAdd = (e) => {
  const files = uploadRef.current.files
  const formData = new FormData()

  for (let i = 0; i < files.length; i++) {
    formData.append("files", files[i]);
  }

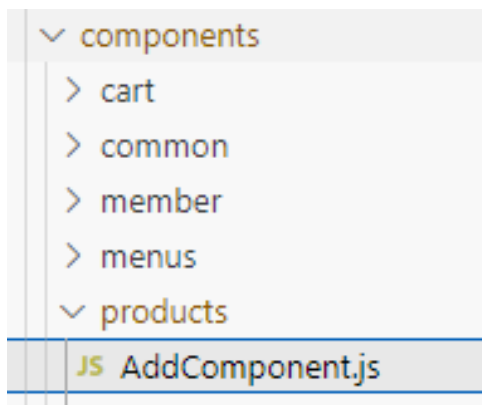
  //other data
  formData.append("pname", product.pname)
  formData.append("pdesc", product.pdesc)
  formData.append("price", product.price)

  addMutation.mutate( formData ) //기존 코드에서 변경
}

const closeModal = () => {
  moveToList({page:1})
}
```

## 장바구니 아이템 컴포넌트

→ components/products/AddComponent.js 수정



```
return (  
  <div className = "border-2 border-sky-200 mt-10 m-2 p-4">  
    <div className="flex justify-center">  
      ...생략 모달 창들은 잠시 삭제  
    </div>  
  </div>  
)  
};  
  
export default AddComponent;
```

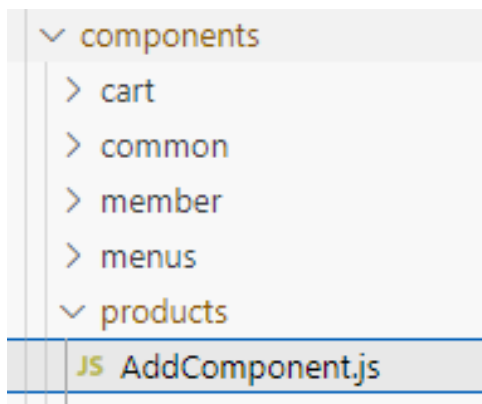
## 장바구니 아이템 컴포넌트

→ useMutation( )의 반환 값 : FetchinModal과 ResultModal을 보이도록 코드 수정

```
return (  
  <div className = "border-2 border-sky-200 mt-10 m-2 p-4">  
  
    {addMutation.isLoading ? <FetchingModal/>: <></>}  
  
    {addMutation.isSuccess ?  
    <ResultModal  
      title={'Add Result'}  
      content={`Add Success ${addMutation.data.result}`}  
      callbackFn={closeModal}/>  
      :  
    <></>  
    }  
  )
```

## 장바구니 아이템 컴포넌트

→ 등록 후 처리 `invalidateQueries()`



```
import { useMutation, useQueryClient } from "@tanstack/react-query";

...생략

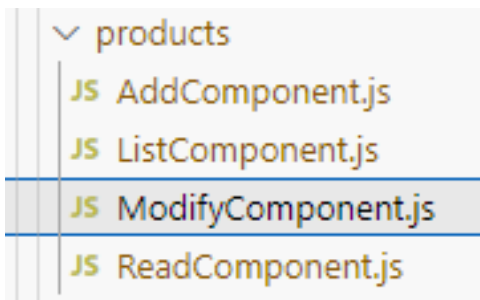
const queryClient = useQueryClient()

const closeModal = () => {
  queryClient.invalidateQueries("products/list")

  moveToList({page:1})
}
```

## 조회 및 상태 처리

→ components/products/ModifyComponent.js 수정



```
import { useQuery } from "@tanstack/react-query";

const ModifyComponent = ({pno}) => {

  const [product, setProduct] = useState(initState)

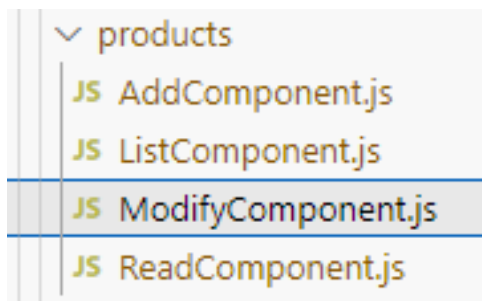
  ...

  const query = useQuery(
    ['products', pno],
    () => getOne(pno) )

  ...생략
```

## 조회 및 상태 처리

→ components/products/ModifyComponent.js 수정



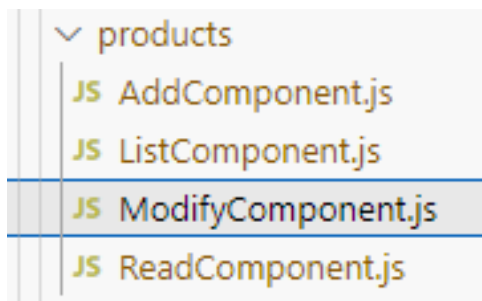
```
const query = useQuery(  
  ['products', pno],  
  () => getOne(pno)  
)  
  
//절대 실행하면 안되는 무한 반복  
if(query.isSuccess) {  
  setProduct(query.data)  
}
```

Uncaught runtime errors:

```
Warning: Too many re-renders. React limits the number of renders to prevent an infinite loop.  
at renderWithHooks (http://localhost:3000/static/js/bundle.js:29600:19)  
at updateFunctionComponent (http://localhost:3000/static/js/bundle.js:32472:24)  
at work (http://localhost:3000/static/js/bundle.js:34184:20)  
at beginWork$1 (http://localhost:3000/static/js/bundle.js:39135:18)  
at renderWithHooks (http://localhost:3000/static/js/bundle.js:29600:19)  
at updateFunctionComponent (http://localhost:3000/static/js/bundle.js:32472:24)  
at beginWork (http://localhost:3000/static/js/bundle.js:34184:20)  
at HTMLUnknownElement.callCallback (http://localhost:3000/static/js/bundle.js:19182:18)  
at Object.invokeGuardedCallbackDev (http://localhost:3000/static/js/bundle.js:19226:20)  
at invokeGuardedCallback (http://localhost:3000/static/js/bundle.js:19283:35)  
at beginWork$1 (http://localhost:3000/static/js/bundle.js:39157:11)  
at performUnitOfWork (http://localhost:3000/static/js/bundle.js:38404:16)  
at workLoopSync (http://localhost:3000/static/js/bundle.js:38327:9)  
at renderRootSync (http://localhost:3000/static/js/bundle.js:38300:11)
```

## 조회 및 상태 처리

→ components/products/ModifyComponent.js 수정



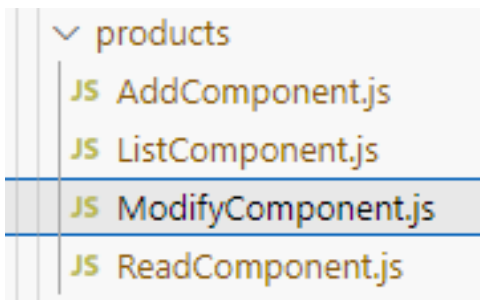
```
const query = useQuery(
  ['products', pno],
  () => getOne(pno),
  {
    staleTime: Infinity
  }
)

useEffect(() => {
  if(query.isSuccess){
    setProduct(query.data)
  }
}, [pno, query.data, query.isSuccess])
```



## 조회 및 상태 처리

→ 삭제 처리 : useMutation을 활용



```
import { useMutation, useQuery, useQueryClient } from "@tanstack/react-query";
import ResultModal from "../common/ResultModal";

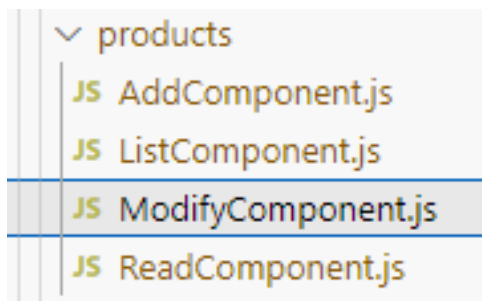
...생략

const delMutation = useMutation((pno) => deleteOne(pno))
const queryClient = useQueryClient()
const handleClickDelete = () => {
  delMutation.mutate(pno)
}

const closeModal = () => {
  if(delMutation.isSuccess) {
    queryClient.invalidateQueries(['products', pno])
    queryClient.invalidateQueries(['products/list'])
    moveToList()
  }
}
```

## 조회 및 상태 처리

→ 삭제 처리 : useMutation을 활용

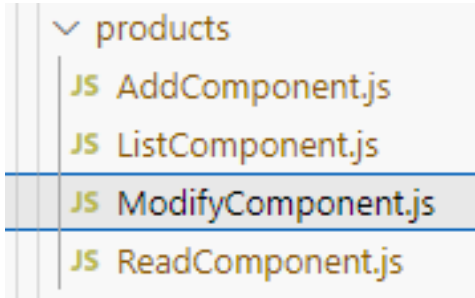


```
return (  
  <div className = "border-2 border-sky-200 mt-10 m-2 p-4">  
    {query.isFetching || delMutation.isLoading ?  
      <FetchingModal/>      :  
      <</>  
    }  
    {  
      delMutation.isSuccess ?  
      <ResultModal  
        title={'처리 결과'}  
        content={'정상적으로 처리되었습니다.'}  
        callbackFn={closeModal}>  
      </ResultModal>  
      :  
      <</>  
    }  
  )  
}
```

...생략

## 조회 및 상태 처리

→ 수정 처리

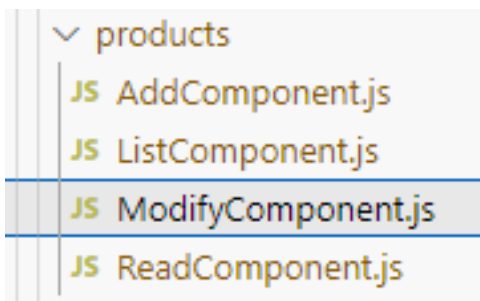


```
const modMutation = useMutation((product) => putOne(pno, product))
const handleClickModify = () => {
  const files = uploadRef.current.files
  const formData = new FormData()
  for (let i = 0; i < files.length; i++) {
    formData.append("files", files[i]);
  }

  //other data
  formData.append("pname", product.pname)
  formData.append("pdesc", product.pdesc)
  formData.append("price", product.price)
  formData.append("delFlag", product.delFlag)
  for( let i = 0; i < product.uploadFileNames.length ; i++){
    formData.append("uploadFileNames", product.uploadFileNames[i])
  }
  modMutation.mutate(formData)
}
```

## 조회 및 상태 처리

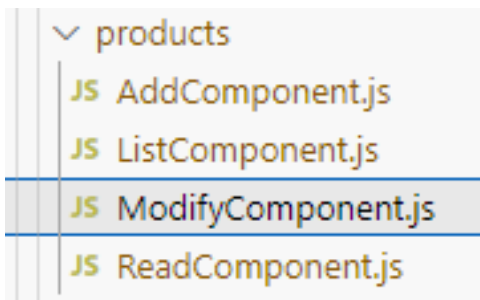
→ 수정 처리



```
const closeModal = () => {  
  
  if(delMutation.isSuccess) {  
    queryClient.invalidateQueries(['products', pno])  
    queryClient.invalidateQueries(['products/list'])  
    moveToList()  
    return  
  }  
  
  if(modMutation.isSuccess) {  
    queryClient.invalidateQueries(['products', pno])  
    queryClient.invalidateQueries(['products/list'])  
    moveToRead(pno)  
  }  
  
}
```

## 조회 및 상태 처리

→ 수정 처리



```
<div className = "border-2 border-sky-200 mt-10 m-2 p-4">

  {query.isFetching || delMutation.isLoading || modMutation.isLoading ?
  <FetchingModal/>
  :
  <></>
  }
  {
    delMutation.isSuccess || modMutation.isSuccess ?
    <ResultModal
      title={'처리 결과'}
      content={'정상적으로 처리되었습니다.'}
      callbackFn={closeModal}>

    </ResultModal>
    :
    <></>
  }
}
```

## 리코일(Recoil) 라이브러리

### → 리코일 라이브러리 소개

리코일은 애플리케이션 내 상태를 처리하는 라이브러리로, 리덕스와 유사한 라이브러리

### → 주요 개념

Atoms : 리코일에서 데이터를 보관하는 역할을 하는 요소.

리코일은 여러 개의 Atoms를 생성하며, 컴포넌트들은 필요한 상태에 선택적으로 접근하여 사용함.

리덕스와는 달리 애플리케이션당 하나의 상태를 유지하는 것이 아닌 여러 개의 Atoms를 사용함.

### → `useRecoilState()`의 활용

`useRecoilState()` : `useState()`의 확장

Atoms를 파라미터로 받아 애플리케이션 내에서 공유되는 데이터에 접근하고 수정할 수 있음.

## 리코일 설치와 설정

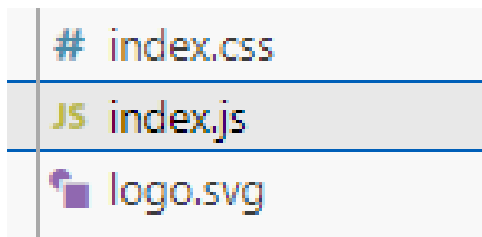
→ 리코일 라이브러리 설치

```
npm install recoil
```

```
"react-cookie": "^6.1.1",  
"react-dom": "^18.2.0",  
"react-redux": "^8.1.3",  
"react-router-dom": "^6.16.0",  
"react-scripts": "5.0.1",  
"recoil": "^0.7.7",  
"web-vitals": "^2.1.4"
```

## 리코일 설치와 설정

→ 리코일 라이브러리 설정



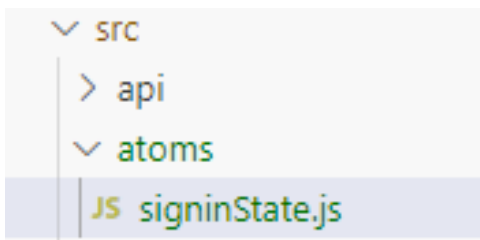
```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { Provider } from 'react-redux';
import store from './store';
import { RecoilRoot } from 'recoil';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Provider store={store}>
    <RecoilRoot>
      <App />
    </RecoilRoot>
  </Provider>
);
```



## 로그인용 Atom

→ atoms 폴더를 생성하고 signInState.js를 추가



```
import { atom } from "recoil";

const initState = {
  email: ''
}

const signInState = atom({
  key: 'signInState',
  default: initState
})

export default signInState
```

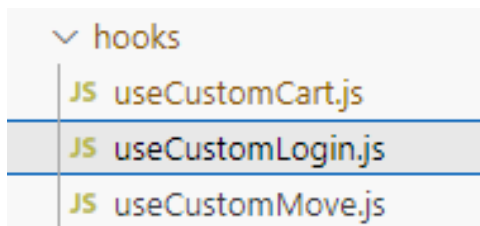
## 로그인용 Atom

→ useRecoilState

- useRecoilValue: 읽기 전용으로 사용
- useSetRecoilState: 쓰기 전용으로 사용
- useResetRecoilState: 초기화 용도

## 로그인용 Atom

→ useCustomLogin내부에서 리덕스 툴킷을 이용했으므로 이를 리코일로 변경

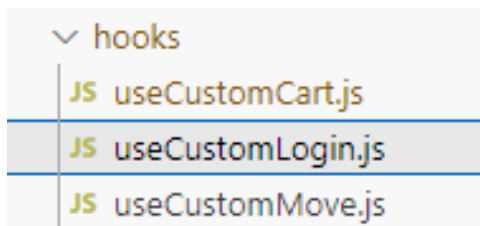


```
import { Navigate, createSearchParams, useNavigate } from "react-router-dom"
import { useRecoilState, useResetRecoilState } from "recoil"
import signinState from "../atoms/signinState"
import { loginPost } from "../api/memberApi"
import { removeCookie, setCookie } from "../util/cookieUtil"

const useCustomLogin = ( ) => {
  const navigate = useNavigate()
  const [loginState, setLoginState] = useRecoilState(signinState)
  const resetState = useResetRecoilState(signinState)
  const isLogin = loginState.email ? true : false //-----로그인 여부
  const doLogin = async (loginParam) => { //-----로그인 함수
    const result = await loginPost(loginParam)
    console.log(result)
    saveAsCookie(result)
    return result
  }
}
```

## 로그인용 Atom

→ useCustomLogin내부에서 리덕스 툴킷을 이용했으므로 이를 리코일로 변경

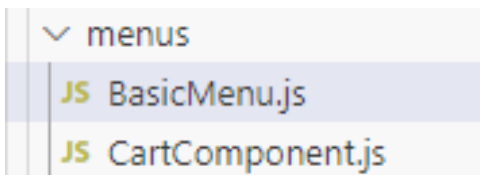


```
const saveAsCookie = (data) => {
  setCookie("member", JSON.stringify(data), 1) //1일
  setLoginState(data)
}
const doLogout = () => { //-----로그아웃 함수
  removeCookie('member')
  resetState()
}
const moveToPath = (path) => { ... }
const moveToLogin = () => { ... }
const moveToLoginReturn = () => { ... }
const exceptionHandle = (ex) => { ... }

//saveAsCookie 추가
return {loginState, isLogin, doLogin, doLogout, saveAsCookie, moveToPath,
moveToLogin, moveToLoginReturn, exceptionHandle}
}
export default useCustomLogin
```

## 로그인용 Atom

→ 상단 메뉴 수정



```
import { Link } from "react-router-dom";
import useCustomLogin from "../../hooks/useCustomLogin";

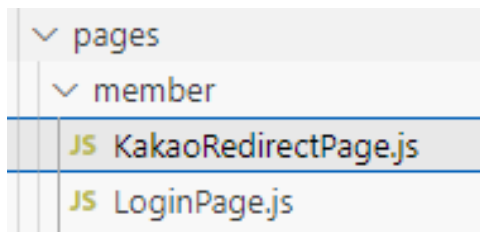
const BasicMenu = () => {

  const {loginState} = useCustomLogin()

  ...생략
```

## 로그인용 Atom

→ 카카오 로그인 처리



```
import { useEffect } from "react";
import { useSearchParams } from "react-router-dom";
import { getAccessToken, getMemberWithAccessToken } from "../../api/kakaoApi";
import useCustomLogin from "../../hooks/useCustomLogin";

const KakaoRedirectPage = () => {
  const [searchParams] = useSearchParams()
  const { moveToPath, saveAsCookie } = useCustomLogin()
  const authCode = searchParams.get("code")
  useEffect(() => {
    getAccessToken(authCode).then(accessToken => {
      console.log(accessToken)
      getMemberWithAccessToken(accessToken).then(memberInfo => {
        console.log("-----")
        console.log(memberInfo)
        saveAsCookie(memberInfo)
        //소셜 회원이 아니라면
        if(memberInfo && !memberInfo.social){
          moveToPath("/")
        }else {
          moveToPath("/member/modify")
        }
      })
    })
  }, [authCode])
  ...이하 생략
```

## 로그인용 Atom

→ 로그인 의 새로 고침 문제



```
import { atom } from "recoil";
import { getCookie } from "../util/cookieUtil";

const initState =
{ email:'',nickname:'',social: false, accessToken:'', refreshToken: ' ' }

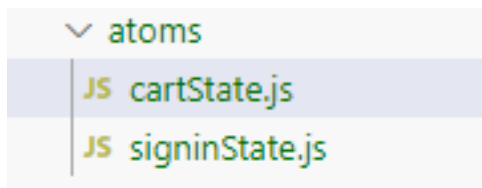
const loadMemberCookie = () => { //쿠키에서 체크
  const memberInfo = getCookie("member")
  //닉네임 처리
  if(memberInfo && memberInfo.nickname) {
    memberInfo.nickname = decodeURIComponent(memberInfo.nickname)
  }
  return memberInfo
}

const signinState = atom({
  key:'signinState', default: loadMemberCookie() || initState
})

export default signinState
```

## 장바구니 처리

→ 장바구니의 아이템에 대한 상태 처리는 리코일을 이용



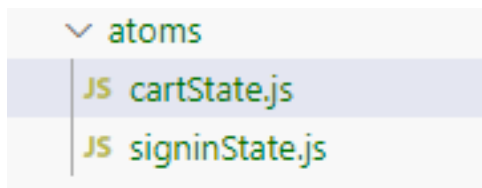
```
import { atom } from "recoil";

export const cartState = atom({
  key: 'cartState',
  default: []
})
```



## 리코일의 Selector

→ Selector는 데이터를 이용해서 처리할 수 있는 기능



```
import { atom, selector } from "recoil";

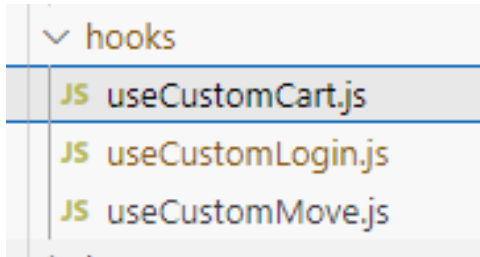
export const cartState = atom({
  key: 'cartState',
  default: []
})

export const cartTotalState = selector( {
  key: "cartTotalState",
  get: ( {get} ) => {
    const arr = get(cartState)
    const initialValue = 0
    const total = arr.reduce((total , current) => total + current.price *
current.qty , initialValue)

    return total
  }
})
```

## 장바구니 데이터 보관

→ 리코일로 만든 cartState를 이용



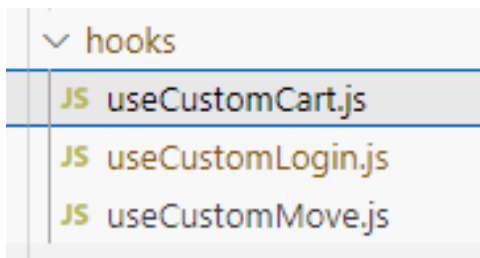
```
import { useMutation, useQuery, useQueryClient } from "@tanstack/react-query"
import { getCartItems, postChangeCart } from "../api/cartApi"
import { useRecoilState } from "recoil"
import { cartState } from "../atoms/cartState"
import { useEffect } from "react"

const useCustomCart = () => {
  const [cartItems, setCartItems] = useRecoilState(cartState)
  const queryClient = useQueryClient()
  const changeMutation = useMutation((param) => postChangeCart(param),
  {onSuccess: (result) => {
    setCartItems(result)
  }})

  // 1 hour
  const query = useQuery(["cart"], getCartItems, {staleTime: 1000 * 60 * 60})
```

## 장바구니 데이터 보관

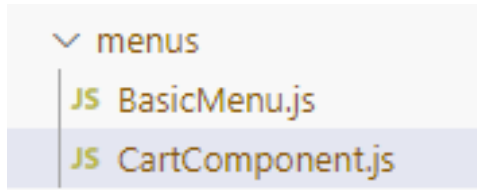
→ 리코일로 만든 cartState를 이용



```
useEffect(() => {  
  if(query.isSuccess) {  
    queryClient.invalidateQueries("cart")  
    setCartItems(query.data)  
  }  
  
}, [query.isSuccess, query.data])  
  
const changeCart = (param) => {  
  changeMutation.mutate(param)  
}  
  
return {cartItems, changeCart}  
  
}  
  
export default useCustomCart
```

## 장바구니 데이터 보관

→ components/menus/CartComponent.js

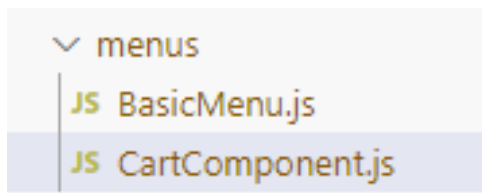


```
import useCustomLogin from "../../hooks/useCustomLogin";
import useCustomCart from "../../hooks/useCustomCart";
import CartItemComponent from "../cart/CartItemComponent";
import { useRecoilValue } from "recoil";
import { cartTotalState } from "../../atoms/cartState";

const CartComponent = () => {
  const { isLogin, loginState } = useCustomLogin()
  const { cartItems, changeCart } = useCustomCart()
  const totalValue = useRecoilValue(cartTotalState)
  return (
    <div className="w-full">
      {isLogin ?
        <div className="flex flex-col">
          <div className="w-full flex">
            <div className="font-extrabold text-2xl w-4/5"> {loginState.nickname}'s Cart
            </div>
            <div className="bg-orange-600 text-center text-white font-bold w-1/5 rounded-full m-1"> {cartItems.length}
            </div>
          </div>
        </div>
      }
    </div>
  )
}
```

## 장바구니 데이터 보관

→ components/menus/CartComponent.js

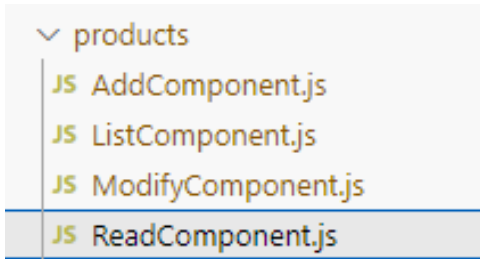


```
<div>
  <ul>
    {cartItems.map( item =>
      <CartItemComponent {...item}
        key={item.cino}
        changeCart={changeCart}
        email={loginState.email}/>)}
    </ul>
  </div>
  <div className="m-2 text-3xl ">
    TOTAL: {totalValue}
  </div>
</div>
:
<div></div>
}
</div>
);
}

export default CartComponent;
```

## 장바구니 아이템 추가

→ useCustomLogin과 useCustomCart를 이용

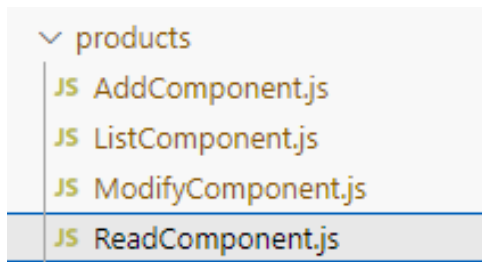


```
import { getOne } from "../../api/productsApi"
import { API_SERVER_HOST } from "../../api/todoApi"
import useCustomCart from "../../hooks/useCustomCart"
import useCustomLogin from "../../hooks/useCustomLogin"
import useCustomMove from "../../hooks/useCustomMove"
import FetchingModal from "../common/FetchingModal"
import { useQuery } from "@tanstack/react-query"

const initState = { ... }
const host = API_SERVER_HOST
const ReadComponent = ({ pno }) => {
  const { moveToList, moveToModify } = useCustomMove()
  const { loginState } = useCustomLogin()
  const { cartItems, changeCart } = useCustomCart()
  const { isFetching, data } = useQuery(
    ['products', pno],
    () => getOne(pno),
    { staleTime: 1000 * 10 * 60, retry: 1 }
  )
}
```

## 장바구니 아이템 추가

→ useCustomLogin과 useCustomCart를 이용



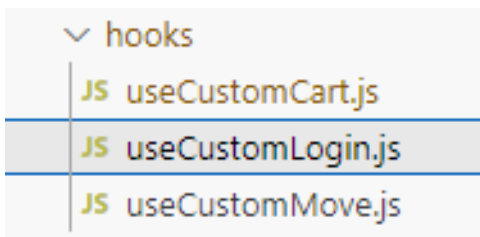
```
const handleClickAddCart = () => {
  let qty = 1
  const addedItem = cartItems.filter(item => item.pno === parseInt(pno))[0]
  if(addedItem) {
    if(window.confirm("이미 추가된 상품입니다. 추가하시겠습니까? ") ===
false) {
      return
    }
    qty = addedItem.qty + 1
  }

  changeCart({email:loginState.email, pno:pno, qty:qty})
}

const product = data || initState
```

## 로그아웃 처리

→ 리코일의 useResetRecoilState를 이용



```
import { Navigate, createSearchParams, useNavigate } from "react-router-dom"
import { useRecoilState, useResetRecoilState } from "recoil"
import signinState from "../atoms/signinState"
import { loginPost } from "../api/memberApi"
import { removeCookie, setCookie } from "../util/cookieUtil"
import { cartState } from "../atoms/cartState"

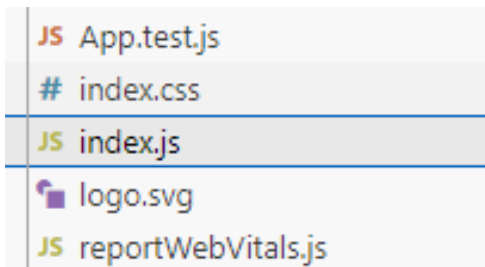
const useCustomLogin = ( ) => {
  const navigate = useNavigate()
  const [loginState, setLoginState] = useRecoilState(signinState)
  const resetState = useResetRecoilState(signinState)
  const resetCartState = useResetRecoilState(cartState) //장바구니 비우기
  ...

  const doLogout = ( ) => { //-----로그아웃 함수
    removeCookie('member')
    resetState()
    resetCartState()
  }
  ...이하 생략
}
```



## 로그아웃 처리

→ 리덕스 툴킷 설정 지우기



```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { RecoilRoot } from 'recoil';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <RecoilRoot>
    <App />
  </RecoilRoot>
);

reportWebVitals();
```

감사합니다.