



YouTube

NAVER 카페

구멍가게 코딩단

# 코드로 배우는 리액트

## 8. 리덕스 툴킷

# 8장. 리덕스 툴킷

- 리액트 컴포넌트는 개별 상태 유지 기능이 필요한데 상태 데이터 공유를 위해 Context나 리덕스 사용
- 개발 용이성을 위해 리덕스 툴킷(Redux Toolkit) 라이브러리 사용
- 사용자 로그인 상태 처리에 활용하고 JWT를 사용하는 API 서버 접근 방식

## 개발목표

1. 리덕스 툴킷 설정
2. 컴포넌트간 상태 공유
3. 비동기 처리와 리덕스 툴킷
4. 쿠키를 이용한 상태 보관

# Index

8.1 리덕스 툴킷 설정

8.2 useSelector/useDispatch

8.3 비동기 호출과 createAsyncThunk( )

8.4 쿠키를 이용한 어플리케이션 상태 저장

8.5 Axios 인터셉터와 Refresh Token

## 리덕스 툴킷 설정

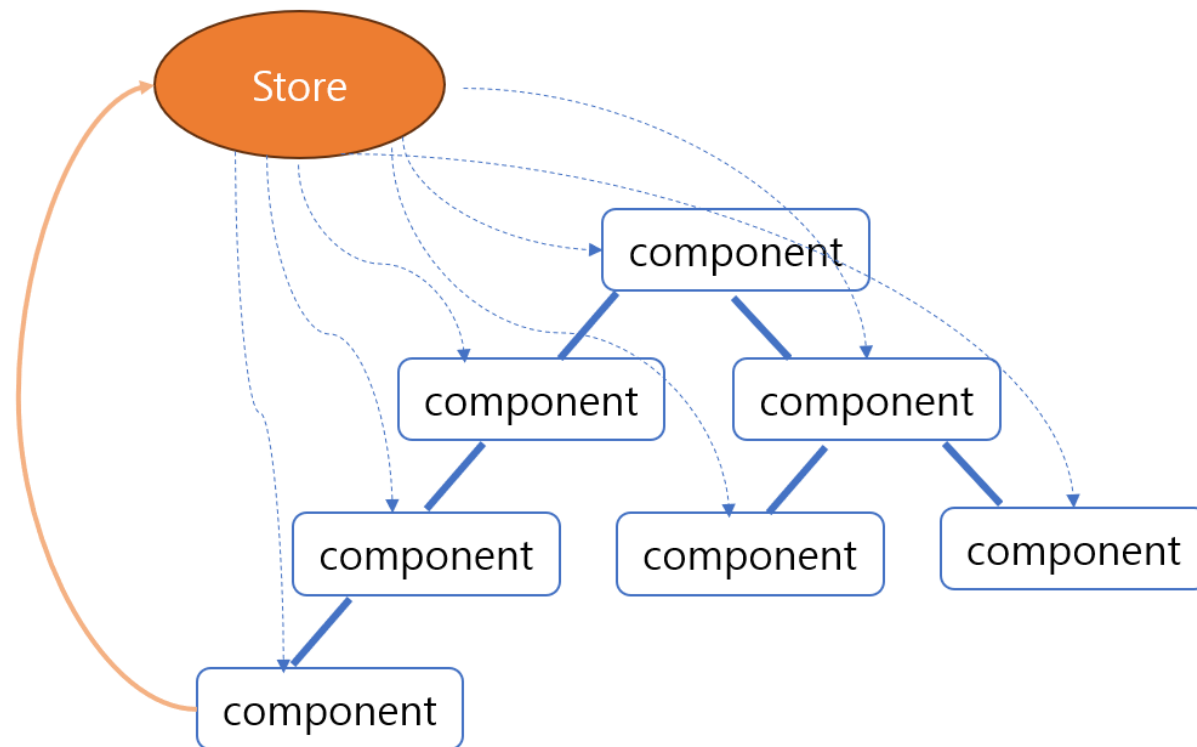
→ @reduxjs/toolkit, react-redux 라이브러리 추가

**'npm install @reduxjs/toolkit react-redux'**

```
"dependencies": {  
  "@reduxjs/toolkit": "^1.9.5",  
  "@testing-library/jest-dom": "^5.16.5",  
  "@testing-library/react": "^13.4.0",  
  "@testing-library/user-event": "^13.5.0",  
  "axios": "^1.4.0",  
  "react": "^18.2.0",  
  "react-dom": "^18.2.0",  
  "react-redux": "^8.1.1",  
  "react-router-dom": "^6.14.1",  
  "react-scripts": "5.0.1",  
  "redux": "^4.2.1",  
  "web-vitals": "^2.1.4"  
},
```

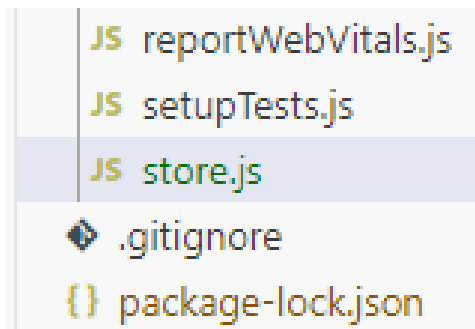
## 스토어 설정

→ '어플리케이션당 하나의 스토어'와 컴포넌트들의 데이터 공유



## 스토어 설정

→ src 폴더내에 store.js 파일을 추가



```
JS reportWebVitals.js
JS setupTests.js
JS store.js
.gitignore
package-lock.json
```

```
import { configureStore } from '@reduxjs/toolkit'

export default configureStore({
  reducer: { }
})
```

## 스토어 설정

→ index.js



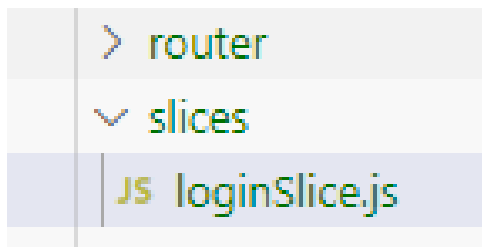
```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { Provider } from 'react-redux';
import store from './store'

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

reportWebVitals();
```

## 슬라이스와 리듀서

→ slices 폴더를 생성, loginSlice.js 작성



```
import { createSlice } from "@reduxjs/toolkit";

const initState = {
  email: ''
}

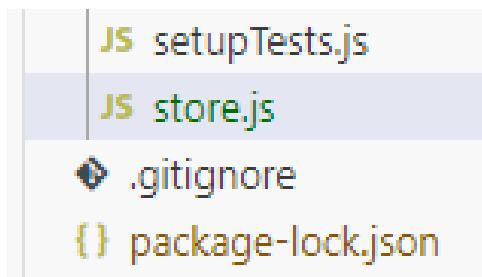
const loginSlice = createSlice({
  name: 'LoginSlice',
  initialState: initState,
  reducers: {
    login: (state, action) => {
      console.log("login.....")
    },
    logout: (state, action) => {
      console.log("logout....")
    }
  }
})

export const {login,logout} = loginSlice.actions
export default loginSlice.reducer
```



## 슬라이스와 리듀서

→ store.js에 slices 설정

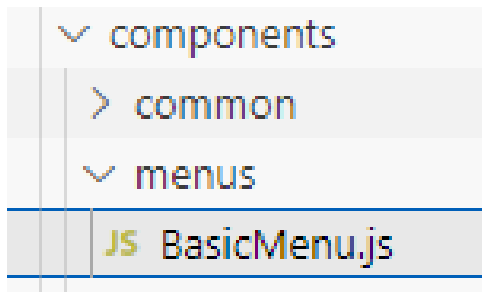


```
import { configureStore } from '@reduxjs/toolkit'
import loginSlice from './slices/loginSlice'

export default configureStore({
  reducer: {
    "loginSlice": loginSlice
  }
})
```

## 로그인 페이지와 로그인

→ BasicMenu.js 컴포넌트



```
import { useSelector } from "react-redux";
import { Link } from "react-router-dom";

const BasicMenu = () => {
  const loginState = useSelector(state => state.loginSlice)

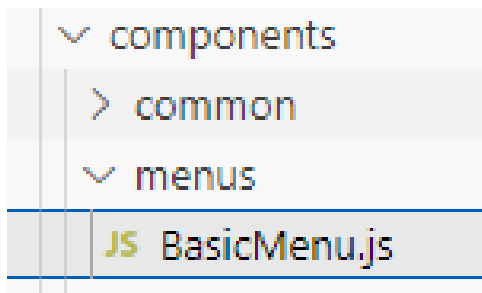
  return (
    <nav id='navbar' className=" flex bg-blue-300">
      ...생략

      <div className="w-1/5 flex justify-end bg-orange-300 p-4 font-medium">
        <div className="text-white text-sm m-1 rounded" >Login</div>
      </div>
    </nav>
  );
}

export default BasicMenu;
```

## 로그인 페이지와 로그인

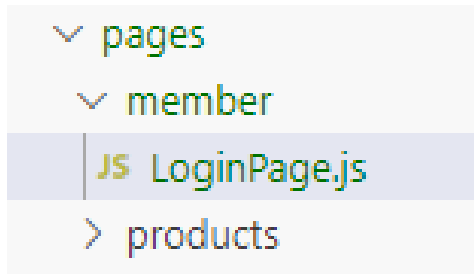
→ BasicMenu.js 컴포넌트



```
<ul className="flex p-4 text-white font-bold">
  <li className="pr-6 text-2xl">
    <Link to={'/'}>Main</Link>
  </li>
  <li className="pr-6 text-2xl">
    <Link to={'/about'}>About</Link>
  </li>
  {loginState.email ? //로그인한 사용자만 출력되는 메뉴
  <>
    <li className="pr-6 text-2xl">
      <Link to={'/todo/'}>Todo</Link>
    </li>
    <li className="pr-6 text-2xl">
      <Link to={'/products/'}>Products</Link>
    </li>
  </>
  :
  <></>
}
</ul>
```

## 로그인 페이지와 로그인

→ pages 폴더내에 member폴더 생성, LoginPage.js파일 추가



```
import BasicMenu from "../../components/menus/BasicMenu";

const LoginPage = () => {
  return (
    <div className='fixed top-0 left-0 z-[1055] flex flex-col h-full w-full'>

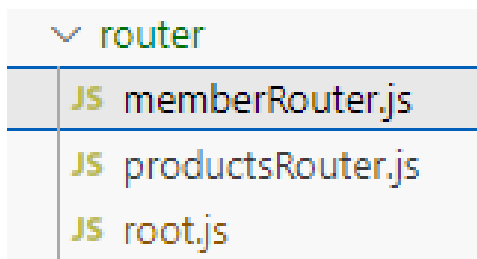
      <BasicMenu/>

      <div className="flex flex-wrap w-full h-full justify-center items-center border-2">
        <div className="text-2xl ">
          Login Page
        </div>
      </div>
    </div>
  );
}

export default LoginPage;
```

## 로그인 페이지와 로그인

→ router/memberRouter.js 경로 설정



```
import { Suspense, lazy } from "react";
const Loading = <div>Loading...</div>
const Login = lazy(() => import("../pages/member/LoginPage"))

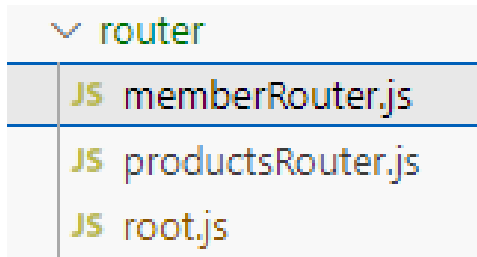
const memberRouter = () => {

  return [
    {
      path: "login",
      element: <Suspense fallback={Loading}><Login/></Suspense>
    }
  ]
}

export default memberRouter
```

## 로그인 페이지와 로그인

→ router/root.js 경로 설정



```
import { Suspense, lazy } from "react";
import todoRouter from "../todoRouter";
import productsRouter from "../productsRouter";
import memberRouter from "../memberRouter";

const { createBrowserRouter } = require("react-router-dom");

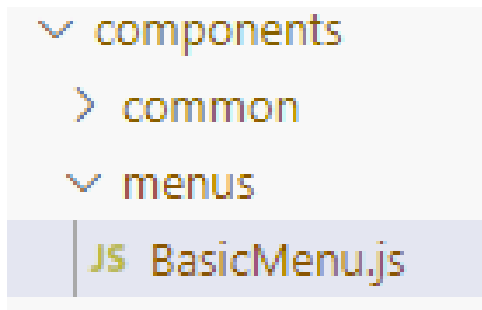
...생략

const root = createBrowserRouter([
  ...생략
  {
    path: "member",
    children: memberRouter()
  }
]);

export default root;
```

## 로그인 페이지와 로그인

→ login 여부에 따른 <Link> 처리



```
<div className="w-1/5 flex justify-end bg-orange-300 p-4 font-medium">
  { ! loginState.email ?

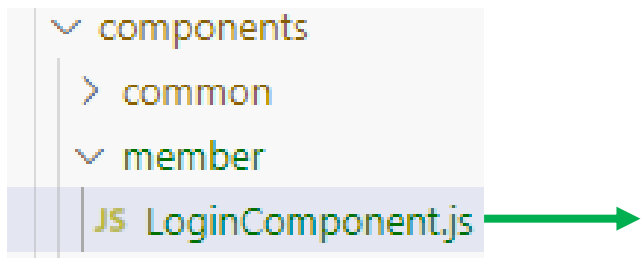
    <div className="text-white text-sm m-1 rounded" >
      <Link to={'/member/login'}>Login</Link>
    </div>
    :

    <>
    </>
  }

</div>
</nav>
```

## 로그인 페이지와 로그인

→ 로그인 컴포넌트 : components/member/LoginComponent.js



```
import { useState } from "react"

const initState = {
  email: '',
  pw: ''
}

const LoginComponent = () => {

  const [loginParam, setLoginParam] = useState({...initState})

  const handleChange = (e) => {
    loginParam[e.target.name] = e.target.value

    setLoginParam({...loginParam})
  }
}
```



## 로그인 컴포넌트

```
return (  
  <div className = "border-2 border-sky-200 mt-10 m-2 p-4">  
    <div className="flex justify-center">  
      <div className="text-4xl m-4 p-4 font-extrabold text-blue-500">Login Component</div>  
    </div>  
    <div className="flex justify-center">  
      <div className="relative mb-4 flex w-full flex-wrap items-stretch">  
        <div className="w-2/5 p-6 text-right font-bold">Email</div>  
        <input className="w-1/5 p-6 rounded-r border border-solid border-neutral-500 shadow-md"  
          name="email" type={'text'} value={loginParam.email} onChange={handleChange} > </input>  
      </div>  
    </div>  
    <div className="flex justify-center">  
      <div className="relative mb-4 flex w-full flex-wrap items-stretch">  
        <div className="w-2/5 p-6 text-right font-bold">Password</div>  
        <input className="w-1/5 p-6 rounded-r border border-solid border-neutral-500 shadow-md"  
          name="pw" type={'password'} value={loginParam.pw} onChange={handleChange} > </input>  
      </div>  
    </div>  
  </div>  
)
```

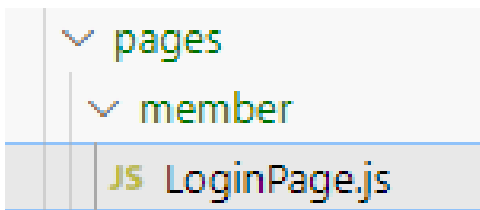
## 로그인 컴포넌트

```
    <div className="flex justify-center">
      <div className="relative mb-4 flex w-full justify-center">
        <div className="w-2/5 p-6 flex justify-center font-bold">
          <button className="rounded p-4 w-36 bg-blue-500 text-xl text-white" `>
            LOGIN
          </button>
        </div>
      </div>
    </div>
  </div>
);
}

export default LoginComponent;
```

## 로그인 페이지와 로그인

→ 로그인 컴포넌트 : LoginPage내부에 LoginComponent 추가



```
import LoginComponent from "../../components/member/LoginComponent";
import BasicMenu from "../../components/menus/BasicMenu";

const LoginPage = () => {
  return (
    <div className='fixed top-0 left-0 z-[1055] flex flex-col h-full w-full'>

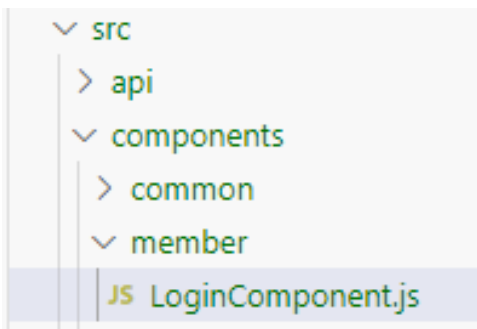
      <BasicMenu/>

      <div className="w-full flex flex-wrap h-full justify-center items-center
border-2">
        <LoginComponent/>
      </div>
    </div>
  );
}

export default LoginPage;
```

## 로그인 페이지와 로그인

→ 로그인 상태 변경 : LoginComponent에 이벤트 처리 함수를 추가, 버튼과 연결



```
import { useState } from "react"
import { useDispatch } from "react-redux"
import { login } from "../../slices/loginSlice"

const initState = {...}

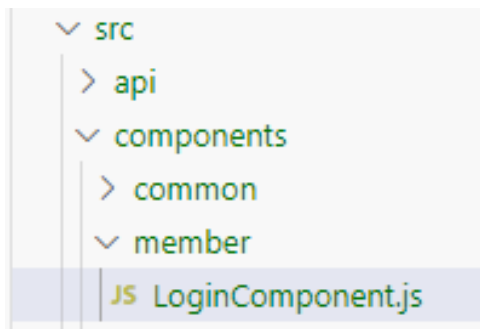
const LoginComponent = () => {

  const [loginParam, setLoginParam] = useState({...initState})
  const dispatch = useDispatch()

  const handleChange = (e) => {
    loginParam[e.target.name] = e.target.value
    setLoginParam({...loginParam})
  }
  const handleClickLogin = (e) => {
    dispatch(login(loginParam))
  }
}
```

## 로그인 페이지와 로그인

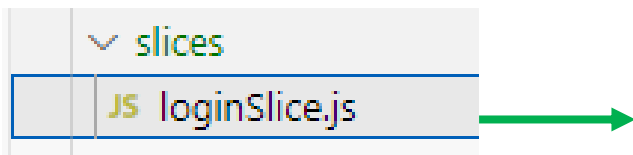
→ 로그인 상태 변경 : LoginComponent에 이벤트 처리 함수를 추가, 버튼과 연결



```
return (  
  <div className = "border-2 border-sky-200 mt-10 m-2 p-4">  
    ...생략  
    <div className="flex justify-center">  
      <div className="relative mb-4 flex w-full justify-center">  
        <div className="w-2/5 p-6 flex justify-center font-bold">  
          <button  
            className="rounded p-4 w-36 bg-blue-500 text-xl text-white"  
            onClick={handleClickLogin} >  
            LOGIN  
          </button>  
        </div>  
      </div>  
    </div>  
  </div>  
}  
export default LoginComponent;
```

## 로그인 페이지와 로그인

→ 로그인 상태 변경 : loginSlice를 이용해서 리듀서가 전달받은 데이터를 확인



```
import { createSlice } from "@reduxjs/toolkit";

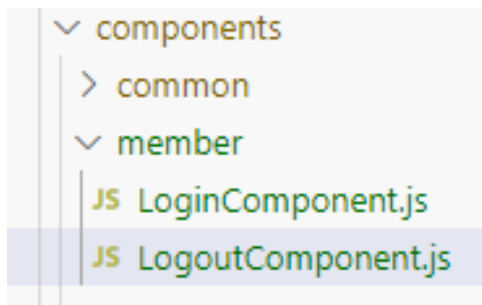
const initState = { email:'' }

const loginSlice = createSlice({
  name: 'loginSlice',
  initialState: initState,
  reducers: {
    login: (state, action) => {
      console.log("login.....")
      const data = action.payload // {email, pw로 구성 }
      return {email: data.email} // 새로운 상태
    },
    logout: (state, action) => { console.log("logout....") }
  }
})

export const {login,logout} = loginSlice.actions
export default loginSlice.reducer
```

## 로그아웃 페이지와 로그아웃

→ pages/member/LogoutPage.js



```
import { useDispatch } from "react-redux";
import { logout } from "../../slices/loginSlice";

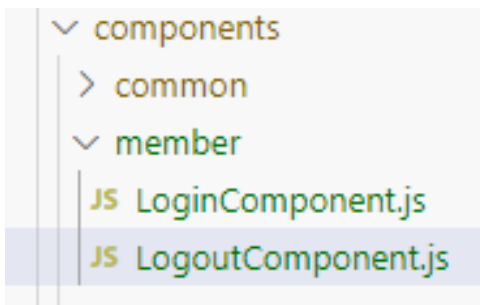
const LogoutComponent = () => {

  const dispatch = useDispatch()

  const handleClickLogout = () => {
    dispatch(logout())
  }
}
```

# 로그아웃 페이지와 로그아웃

→ pages/member/LogoutPage.js



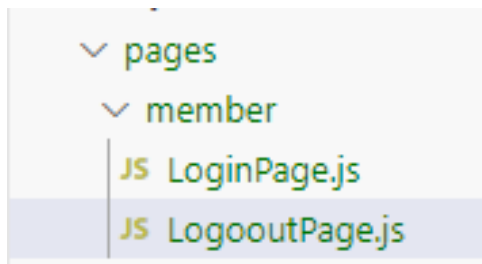
```
return (
  <div className = "border-2 border-red-200 mt-10 m-2 p-4">
    <div className="flex justify-center">
      <div className="text-4xl m-4 p-4 font-extrabold text-red-500">
        Logout Component
      </div>
    </div>
    <div className="flex justify-center">
      <div className="relative mb-4 flex w-full justify-center">
        <div className="w-2/5 p-6 flex justify-center font-bold">
          <button className="rounded p-4 w-36 bg-red-500 text-xl text-white"
            onClick={handleClickLogout} > LOGOUT </button>
        </div>
      </div>
    </div>
  </div>
);
}

export default LogoutComponent;
```



## 로그아웃 페이지와 로그아웃

→ components/member/LogoutComponent.js



```
import LogoutComponent from "../../components/member/LogoutComponent";
import BasicMenu from "../../components/menus/BasicMenu";

const LogoutPage = () => {
  return (
    <div className='fixed top-0 left-0 z-[1055] flex flex-col h-full w-full'>

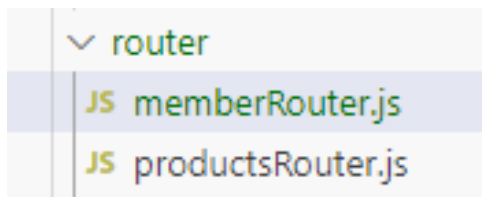
      <BasicMenu/>

      <div className="w-full flex flex-wrap h-full justify-center items-
center border-2">
        <LogoutComponent/></LogoutComponent>
      </div>
    </div>
  );
}

export default LogoutPage;
```

## 로그아웃 페이지와 로그아웃

→ LoginPage를 라우팅 : memberRouter.js 수정



```
import { Suspense, lazy } from "react";
...

const LoginPage = lazy(() => import("../pages/member/LogoutPage"))

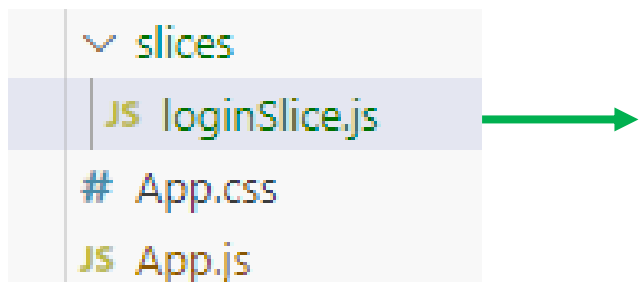
const memberRouter = () => {

  return [
    ...생략
    {
      path: "logout",
      element: <Suspense fallback={Loading}><LogoutPage/></Suspense>,
    }
  ]
}

export default memberRouter
```

## 로그아웃 페이지와 로그아웃

→ 로그아웃 상태 변경 : loginSlice.js 수정



```
import { createSlice } from "@reduxjs/toolkit";

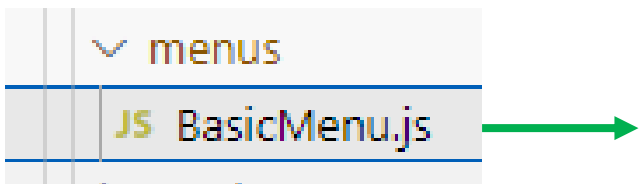
const initState = { email:'' }

const loginSlice = createSlice({
  name: 'loginSlice',
  initialState: initState,
  reducers: {
    login: (state, action) => {      ...생략      },
    logout: (state, action) => {
      console.log("logout....")
      return {...initState}
    }
  }
})

export const {login,logout} = loginSlice.actions
export default loginSlice.reducer
```

## 로그아웃 페이지와 로그아웃

→ 로그아웃 상태 변경 : 화면 상단의 메뉴에 로그인된 상황에서는 로그아웃 링크를 추가



```
<div className="w-1/5 flex justify-end bg-orange-300 p-4 font-medium">
  { ! loginState.email ?

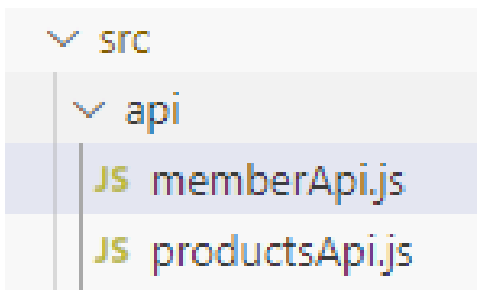
    <div className="text-white text-sm m-1 rounded" >
      <Link to={'/member/login'}>Login</Link>
    </div>
    :

    <div className="text-white text-sm m-1 rounded" >
      <Link to={'/member/logout'}>Logout</Link>
    </div>

  }
</div>
```

## 비동기 호출과 createAsyncThunk( )

→ api폴더내에 memberApi.js파일을 추가



```
import axios from "axios"
import { API_SERVER_HOST } from "../todoApi"

const host = `${API_SERVER_HOST}/api/member`

export const loginPost = async (loginParam) => {

  const header = {headers: {"Content-Type": "x-www-form-urlencoded"}}

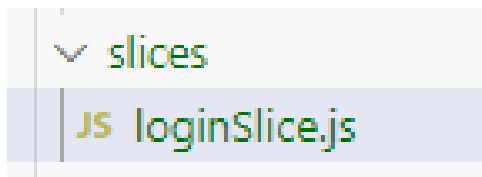
  const form = new FormData()
  form.append('username', loginParam.email)
  form.append('password', loginParam.pw)

  const res = await axios.post(`${host}/login`, form, header)

  return res.data
}
```

## 비동기 호출과 createAsyncThunk( )

→ api폴더내에 memberApi.js파일을 추가



```
import { createAsyncThunk, createSlice } from "@reduxjs/toolkit";
import { loginPost } from "../api/memberApi";

const initState = { email: '' }

export const loginPostAsync = createAsyncThunk('loginPostAsync', (param) => {
  return loginPost(param)
})

const loginSlice = createSlice({
  name: 'LoginSlice',
  initialState: initState,
  reducers: {
    login: (state, action) => {
      console.log("login.....")
      const data = action.payload // {email, pw로 구성 }
      return {email: data.email} // 새로운 상태
    },
  },
})
```

## 비동기 호출과 createAsyncThunk( )

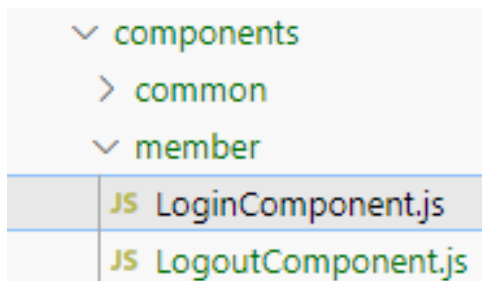
→ api폴더내에 memberApi.js파일을 추가



```
logout: (state, action) => {
  console.log("logout...")
  return {...initState}
},
extraReducers: (builder) => {
  builder.addCase(loginPostAsync.fulfilled, (state, action) => {
    console.log("fulfilled")
  })
  .addCase(loginPostAsync.pending, (state, action) => {
    console.log("pending")
  })
  .addCase(loginPostAsync.rejected, (state, action) => {
    console.log("rejected")
  })
}
})
export const {login, logout} = loginSlice.actions
export default loginSlice.reducer
```

## 비동기 호출과 createAsyncThunk( )

→ '/api/member/login'을 호출하도록 LoginComponent를 수정



```
import { useState } from "react"
import { useDispatch } from "react-redux"
import { loginPostAsync } from "../../slices/loginSlice"

const initState = {...}

const LoginComponent = () => {

  ...생략

  const handleClickLogin = (e) => {

    //dispatch(login(loginParam)) //동기화된 호출
    //비동기 호출
    dispatch( loginPostAsync(loginParam) ) // loginSlice의 비동기 호출

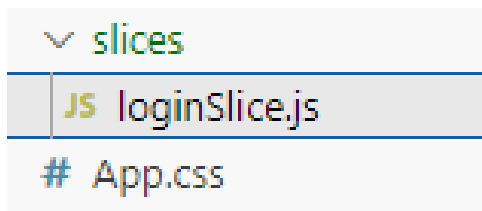
  }

  ...생략
```



## 로그인 후처리

→ API 서버에서 로그인시에 전송되는 데이터들을 상태데이터로 보관



```
...생략
extraReducers: (builder) => {

  builder.addCase( loginPostAsync.fulfilled, (state, action) => {
    console.log("fulfilled")

    const payload = action.payload
    return payload
  })

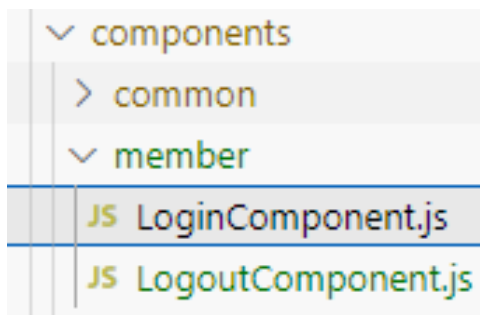
  .addCase(loginPostAsync.pending, (state, action) => {
    console.log("pending")
  })

  .addCase(loginPostAsync.rejected, (state, action) => {
    console.log("rejected")
  })

}
...생략
```

## 로그인 후처리

→ unwrap( )을 이용한 후처리



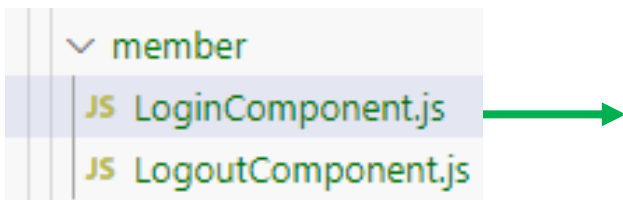
```
const handleClickLogin = (e) => {

  //dispatch(login(loginParam)) //동기화된 호출
  //비동기 호출
  dispatch( loginPostAsync(loginParam) ) // loginSlice의 비동기 호출
  .unwrap()
  .then(data => {
    console.log("after unwrap....")
    console.log(data)
  })

}
```

## 로그인 후처리

→ unwrap( )을 이용한 후처리 : 로그인 결과에 맞게 경고창을 보이도록 수정



```
const handleClickLogin = (e) => {  
  
  dispatch( loginPostAsync(loginParam) )  
    .unwrap()  
    .then(data => {  
      console.log("after unwrap....")  
      console.log(data)  
      if(data.error) {  
        alert("이메일과 패스워드를 다시 확인하세요")  
      }else {  
        alert("로그인 성공")  
      }  
    })  
}
```

## 로그인 후 처리

→ 로그인 후 이동 처리 : : LoginComponent에서 정상적으로 로그인 후에 이동

▼ member  
JS LoginComponent.js  
JS LogoutComponent.js



```
import { useNavigate } from "react-router-dom"

const initState = { email: '', pw: '' }

const LoginComponent = () => {

  const [loginParam, setLoginParam] = useState({...initState})

  const navigate = useNavigate()

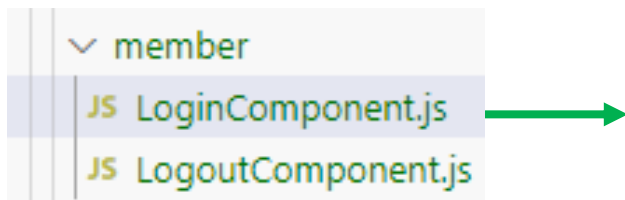
  const dispatch = useDispatch()

  const handleChange = (e) => {
    loginParam[e.target.name] = e.target.value

    setLoginParam({...loginParam})
  }
}
```

## 로그인 후 처리

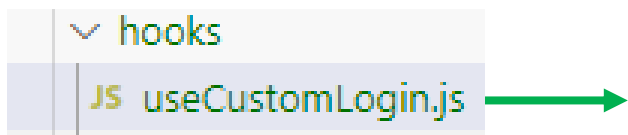
→ 로그인 후 이동 처리 : LoginComponent에서 정상적으로 로그인 후에 이동



```
const handleClickLogin = (e) => {  
  
  //dispatch(login(loginParam)) //동기화된 호출  
  //비동기 호출  
  dispatch( loginPostAsync(loginParam) ) // loginSlice의 비동기 호출  
  .unwrap()  
  .then(data => {  
    console.log("after unwrap....")  
    if(data.error) {  
      alert("이메일과 패스워드를 다시 확인하세요")  
    }else {  
      alert("로그인 성공")  
      //뒤로 가기 했을 때 로그인 화면을 볼 수 없게  
      navigate({pathname: `/`}, {replace:true})  
    }  
  })  
})  
}  
...생략
```

## 로그인 후처리

→ 로그인 관련 기능 처리를 위한 커스텀 훅



```
import { useDispatch, useSelector } from "react-redux"
import { Navigate, useNavigate } from "react-router-dom"
import { loginPostAsync, logout } from "../slices/loginSlice"

const useCustomLogin = ( ) => {

  const navigate = useNavigate()

  const dispatch = useDispatch()

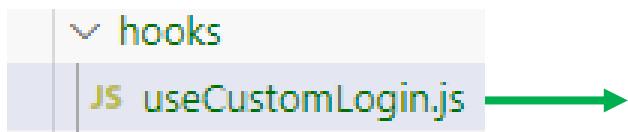
  const loginState = useSelector(state => state.loginSlice) //---로그인 상태

  const isLogin = loginState.email ? true : false //-----로그인 여부

  const doLogin = async (loginParam) => { //-----로그인 함수
    const action = await dispatch(loginPostAsync(loginParam))
    return action.payload
  }
}
```

## 로그인 후처리

→ 로그인 관련 기능 처리를 위한 커스텀 훅



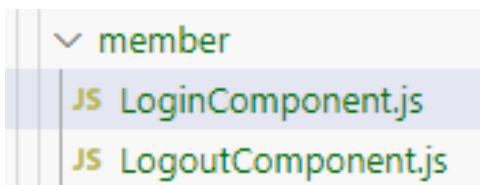
```
const doLogout = () => { //-----로그아웃 함수
  dispatch(logout())
}
const moveToPath = (path) => { //-----페이지 이동
  navigate({pathname: path}, {replace:true})
}
const moveToLogin = () => { //-----로그인 페이지로 이동
  navigate({pathname: '/member/login'}, {replace:true})
}
const moveToLoginReturn = () => { //-----로그인 페이지로 이동 컴포넌트
  return <Navigate replace to="/member/login"/>
}

return {loginState, isLogin, doLogin, doLogout, moveToPath, moveToLogin,
moveToLoginReturn}
}

export default useCustomLogin
```

## 로그인 후처리

→ 로그인 관련 기능 처리를 위한 커스텀 훅 : LoginComponent의 수정



```
import { useState } from "react"
import useCustomLogin from "../../hooks/useCustomLogin"

const initState = {
  email: '',
  pw: ''
}

const LoginComponent = () => {

  const [loginParam, setLoginParam] = useState({...initState})

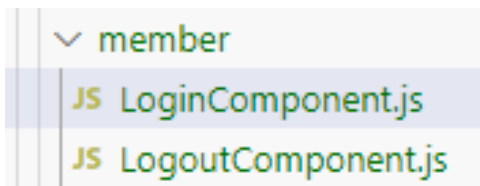
  const {doLogin, moveToPath} = useCustomLogin()

  const handleChange = (e) => {
    loginParam[e.target.name] = e.target.value
    setLoginParam({...loginParam})
  }
}
```



## 로그인 후처리

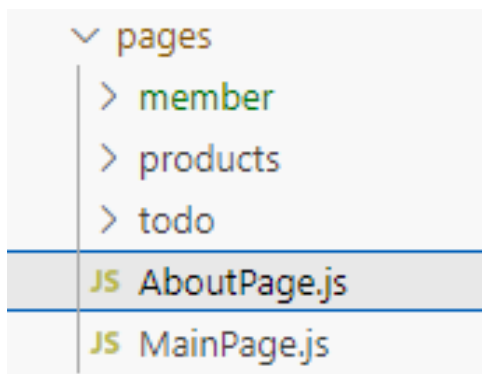
→ 로그인 관련 기능 처리를 위한 커스텀 훅 : LoginComponent의 수정



```
const handleClickLogin = (e) => {  
  
  doLogin(loginParam) // loginSlice의 비동기 호출  
  .then(data => {  
    console.log(data)  
  
    if(data.error) {  
      alert("이메일과 패스워드를 다시 확인하세요")  
    }else {  
      alert("로그인 성공")  
      moveToPath('/')  
    }  
  })  
}  
...생략
```

## 로그인이 필요한 페이지

→ useCustomLogin을 이용한 로그인 체크



```
import useCustomLogin from "../hooks/useCustomLogin";
import BasicLayout from "../layouts/BasicLayout";

const AboutPage = () => {
  const {isLogin, moveToLoginReturn} = useCustomLogin()

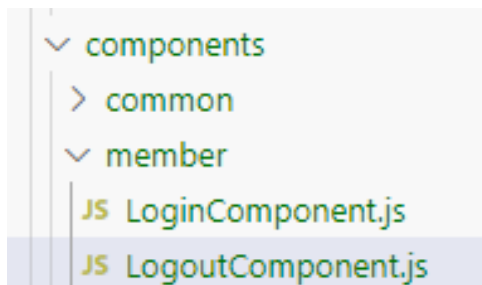
  if(!isLogin){
    return moveToLoginReturn()
  }
  return (
    <BasicLayout>
      <div className=" text-3xl">About Page</div>
    </BasicLayout>

  );
}

export default AboutPage;
```

## 로그 아웃 처리

→ loginSlice의 logout( )을 그대로 활용



```
import useCustomLogin from "../../hooks/useCustomLogin";

const LogoutComponent = () => {

  const {doLogout, moveToPath} = useCustomLogin()

  const handleClickLogout = () => {
    doLogout()
    alert("로그아웃되었습니다.")
    moveToPath("/")
  }

  return (
    ...생략
  );
}

export default LogoutComponent;
```

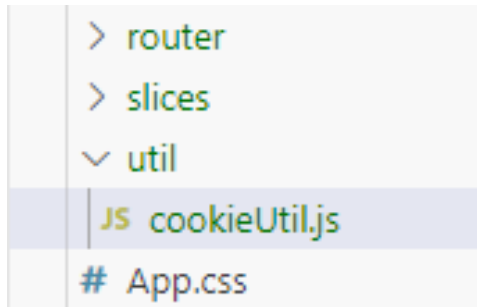
## 쿠키를 이용한 어플리케이션 상태 저장

→ '.npm install'등을 이용해서 'react-cookie'를 추가

```
PS C:\Users\cooki\frontend\mall> npm install react-cookie  
  
added 3 packages, and audited 1528 packages in 3s  
  
245 packages are looking for funding  
  run `npm fund` for details
```

## 쿠키를 이용한 어플리케이션 상태 저장

→ util 폴더를 추가하고 cookieUtil.js파일을 추가



```
import { Cookies } from "react-cookie";

const cookies = new Cookies()

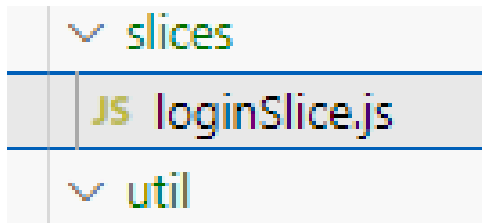
export const setCookie = (name, value, days) => {
  const expires = new Date()
  expires.setUTCDate(expires.getUTCDate() + days ) //보관기한
  return cookies.set(name, value, {path: '/', expires:expires})
}

export const getCookie = (name) => {
  return cookies.get(name)
}

export const removeCookie = (name , path="/") => {
  cookies.remove(name, {path} )
}
```

## 로그인 결과의 쿠키 보관

→ 로그인에 성공하면 결과를 쿠키로 보관하는 처리 : loginSlice에서 cookieUtil을 이용하도록 수정



```
...생략
import { setCookie } from "../util/cookieUtil";

...생략
builder.addCase( loginPostAsync.fulfilled, (state, action) => {
  console.log("fulfilled")

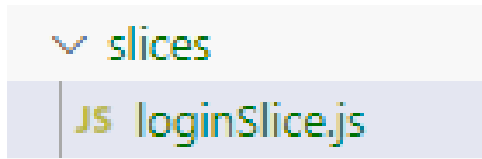
  const payload = action.payload

  //정상적인 로그인시에만 저장
  if(!payload.error){
    setCookie( " member " ,JSON.stringify(payload), 1) //1일
  }

  return action.payload
})
```

## 로그인 결과의 쿠키 보관

→ 어플리케이션 로딩시 쿠키 활용



```
import { createAsyncThunk, createSlice } from "@reduxjs/toolkit";
import { loginPost } from "../api/memberApi";

import { getCookie, setCookie } from "../util/cookieUtil";

const initState = { email:'' }
const loadMemberCookie = () => { //쿠키에서 로그인 정보 로딩

  const memberInfo = getCookie("member")

  //닉네임 처리
  if(memberInfo && memberInfo.nickname) {
    memberInfo.nickname = decodeURIComponent(memberInfo.nickname)
  }

  return memberInfo
}
```

## 로그인 결과의 쿠키 보관

→ 어플리케이션 로딩시 쿠키 활용

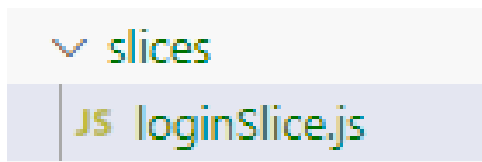


```
export const loginPostAsync = createAsyncThunk('loginPostAsync', (param) => {  
  return loginPost(param)  
})  
  
const loginSlice = createSlice({  
  name: 'LoginSlice',  
  initialState: loadMemberCookie() || initState, //쿠키가 없다면 초깃값사용  
  ...생략  
})  
  
export const {login,logout} = loginSlice.actions  
  
export default loginSlice.reducer
```



## 로그인 결과의 쿠키 보관

→ 로그아웃의 쿠키 처리



```
import { getCookie, removeCookie, setCookie } from "../util/cookieUtil";

...생략

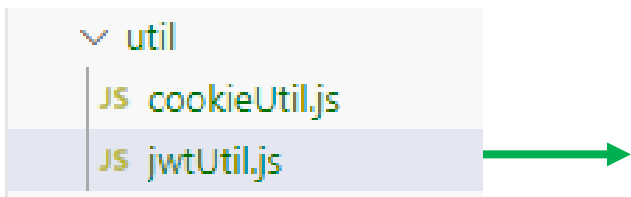
logout: (state, action) => {
  console.log("logout...")

  removeCookie("member")

  return {...initState}
}
```

## Axios 인터셉터와 Refresh Token

→ util 패키지에 jwtUtil.js파일을 추가



```
import axios from "axios";

const jwtAxios = axios.create()

//before request
const beforeReq = (config) => {
  console.log("before request.....")
  return config
}

//fail request
const requestFail = (err) => {
  console.log("request error.....")
  return Promise.reject(err)
}

//before return response
const beforeRes = async (res) => {
  console.log("before return response.....")
  return res
}

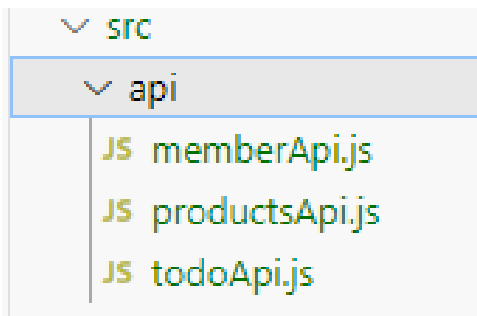
//fail response
const responseFail = (err) => {
  console.log("response fail error.....")
  return Promise.reject(err);
}

jwtAxios.interceptors.request.use( beforeReq, requestFail )
jwtAxios.interceptors.response.use( beforeRes, responseFail )

export default jwtAxios
```

## Axios 인터셉터와 Refresh Token

→ 기존의 axios 대신 memberApi.js와 productsApi.js에서 jwtAxios를 이용하도록 변경



```
import jwtAxios from "../util/jwtUtil"

...생략

export const getOne = async (tno) => {

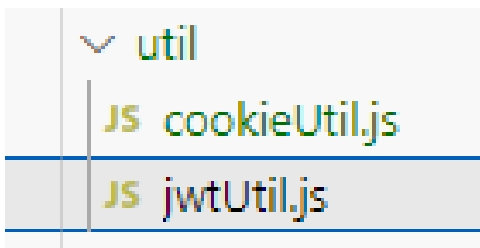
  const res = await jwtAxios.get(`${prefix}/${tno}` )

  return res.data

}
```

## Access Token의 전달

→ Access Token의 전달



```
import axios from "axios";
import { getCookie } from "../cookieUtil";

const jwtAxios = axios.create()

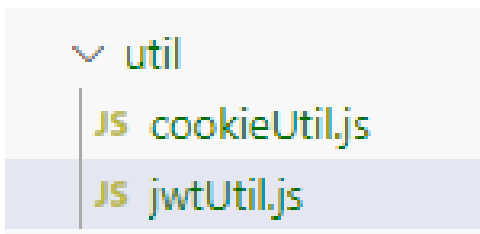
//before request
const beforeReq = (config) => {
  console.log("before request.....")
  const memberInfo = getCookie("member")
  if( !memberInfo ) {
    console.log("Member NOT FOUND")
    return Promise.reject(
      {response:
        {data:
          {error:"REQUIRE_LOGIN"}
        }
      }
    )
  }

  const {accessToken} = memberInfo
  // Authorization 헤더 처리
  config.headers.Authorization = `Bearer ${accessToken}`
  return config
}

...생략
```

## Access Token의 전달

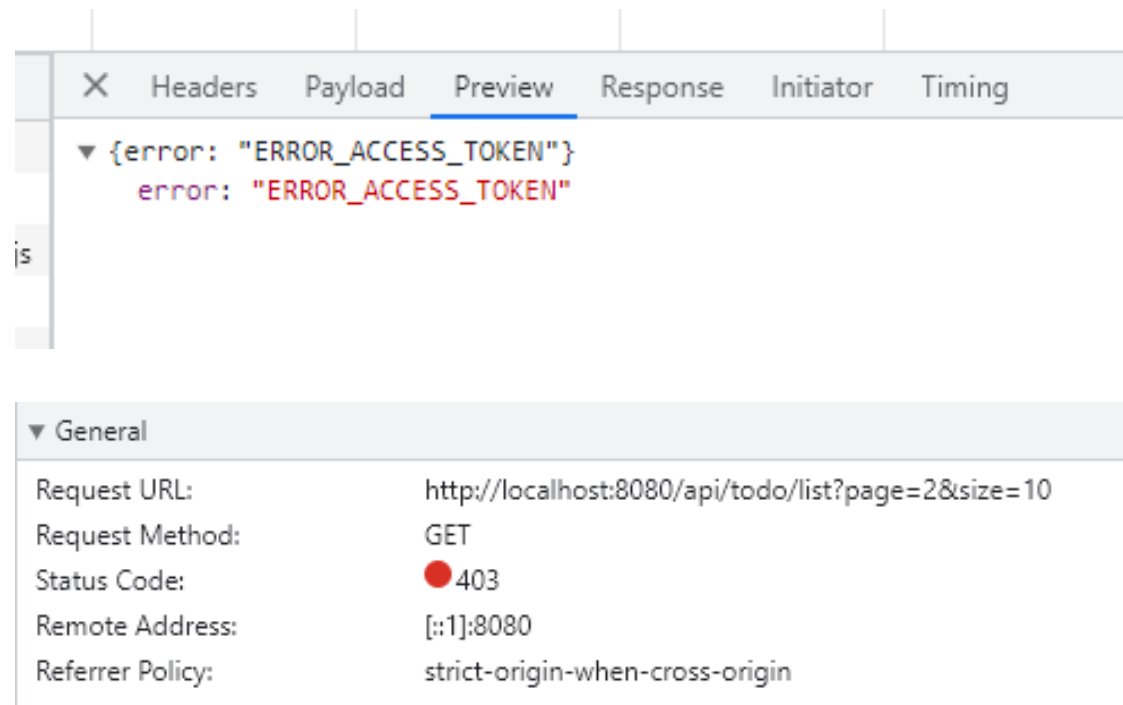
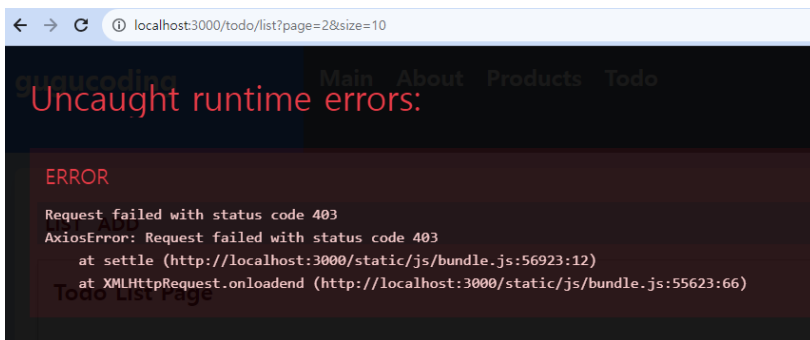
→ jwtUtil.js에서 Access Token을 API 서버 호출 전에 Authorization 헤더를 추가하도록 구성



```
const {accessToken} = memberInfo
// Authorization 헤더 처리
config.headers.Authorization = `Bearer ${accessToken}`
return config
}
...생략
```

## Access Token의 전달

→ 유효시간이 지난 Access Token : API 서버 호출 시 10분이 지난 후에는 만료된 Access Token



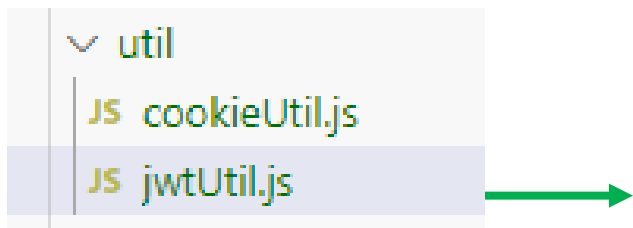
## Refresh Token을 이용한 자동 갱신

→ 사일런트 리프레시

Access Token의 만료 후, Refresh Token을 활용하여 자동으로 새로운 Access Token을 갱신하는 방식

## Refresh Token을 이용한 자동 갱신

→ jwtUtil.js에 Access Token, Refresh Token을 이용해서 '/api/member/refresh'를 호출



```
//before return response
const beforeRes = async (res) => {
  console.log("before return response.....")
  console.log(res)
  const data = res.data

  if(data && data.error === 'ERROR_ACCESS_TOKEN'){
    const memberCookieValue = getCookie("member")
    const result = await refreshJWT( memberCookieValue.accessToken,
    memberCookieValue.refreshToken )
    console.log("refreshJWT RESULT", result)

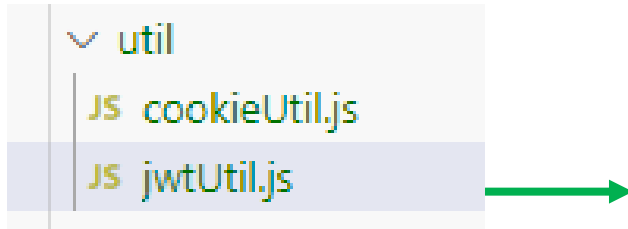
    memberCookieValue.accessToken = result.accessToken
    memberCookieValue.refreshToken = result.refreshToken

    setCookie("member", JSON.stringify(memberCookieValue), 1)
  }
  return res
}
```



## Refresh Token을 이용한 자동 갱신

→ 갱신된 토큰의 저장과 재호출



```
import axios from "axios";
import { getCookie, setCookie } from "../cookieUtil";
import { API_SERVER_HOST } from "../api/todoApi";

...생략

//before return response
const beforeRes = async (res) => {
  console.log("before return response.....")

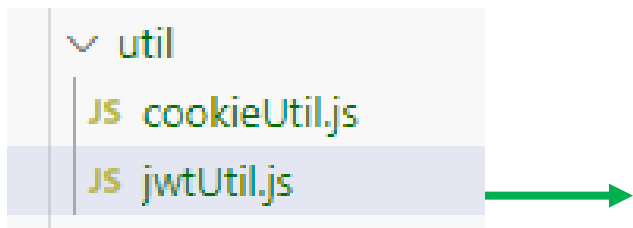
  //console.log(res)
  //'ERROR_ACCESS_TOKEN'
  const data = res.data

  if(data && data.error === 'ERROR_ACCESS_TOKEN'){

    const memberCookieValue = getCookie("member")
```

## Refresh Token을 이용한 자동 갱신

→ 갱신된 토큰의 저장과 재호출



```
const result = await refreshJWT( memberCookieValue.accessToken,
memberCookieValue.refreshToken )
console.log("refreshJWT RESULT", result)

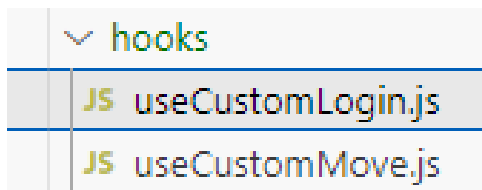
memberCookieValue.accessToken = result.accessToken
memberCookieValue.refreshToken = result.refreshToken

setCookie("member", JSON.stringify(memberCookieValue), 1)

//원래의 호출
const originalRequest = res.config
originalRequest.headers.Authorization = `Bearer ${result.accessToken}`
return await axios(originalRequest)
}
return res
}
```

## Refresh Token을 이용한 자동 갱신

→ 토큰에 따른 예외 처리



```
import { useDispatch, useSelector } from "react-redux"
import { Navigate, createSearchParams, useNavigate } from "react-router-dom"
import { loginPostAsync, logout } from "../slices/loginSlice"

const useCustomLogin = ( ) => {
  ...생략

  const exceptionHandle = (ex) => {
    console.log("Exception-----")
    console.log(ex)

    const errorMsg = ex.response.data.error
    const errorStr = createSearchParams({error: errorMsg}).toString()

    if(errorMsg === 'REQUIRE_LOGIN'){
      alert("로그인 해야만 합니다.")
      navigate({pathname: '/member/login' , search: errorStr})
      return
    }
  }
```

## Refresh Token을 이용한 자동 갱신

→ 토큰에 따른 예외 처리

hooks
JS useCustomLogin.js
JS useCustomMove.js



```
if(ex.response.data.error === 'ERROR_ACCESSDENIED'){
  alert("해당 메뉴를 사용할 수 있는 권한이 없습니다.")
  navigate({pathname: '/member/login' , search: errorStr})
  return
}

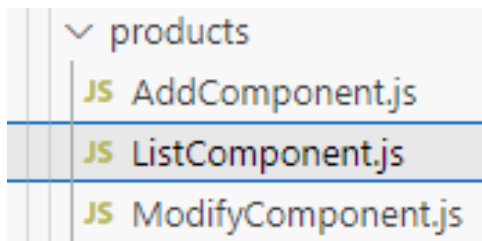
return {loginState, isLogin, doLogin, doLogout, moveToPath, moveToLogin,
moveToLoginReturn, exceptionHandle}

}

export default useCustomLogin
```

## Refresh Token을 이용한 자동 갱신

→ 토큰에 따른 예외 처리

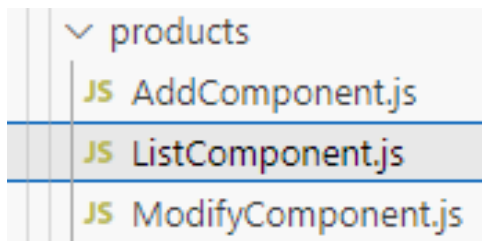


```
import { useEffect, useState } from "react";
import { getList } from "../../api/productsApi";
import useCustomMove from "../../hooks/useCustomMove";
import FetchingModal from "../../common/FetchingModal";
import { API_SERVER_HOST } from "../../api/todoApi";
import PageComponent from "../../common/PageComponent";
import useCustomLogin from "../../hooks/useCustomLogin";

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

## Refresh Token을 이용한 자동 갱신

→ 토큰에 따른 예외 처리



```
const host = API_SERVER_HOST

const ListComponent = () => {
  const {exceptionHandle} = useCustomLogin()
  const {page, size, refresh, moveToList, moveToRead} = useCustomMove()
  //serverData는 나중에 사용
  const [serverData, setServerData] = useState(initState)
  //for FetchingModal
  const [fetching, setFetching] = useState(false)
  useEffect(() => {
    setFetching(true)
    getList({page, size}).then(data => {
      console.log(data)
      setServerData(data)
      setFetching(false)
    }).catch(err => exceptionHandle(err))
  }, [page, size, refresh])
  ...생략
```

감사합니다.