YouTube
NAVER 카페
구멍가게 코딩단

코드로 배우는 리액트

2. React-Router 설정

2장. React-Router 설정

IA(Information Architecture) 기획에 맞는 메뉴 구성

React-Router

- 정보 구조 기획(IA)이 웹 사이트 개발 중요 단계.
- IA는 '메뉴 경로' 정리로 페이지 간 링크 및 <form> 설계.
- 리액트는 SPA로 동작, 여러 컴포넌트로 하나의 페이지에 표시.
- React-Router로 브라우저 주소창에 따라 다른 화면 표시.

개발목표

1. React-Router를 적용해서 페이지의 이동이 가능하도록 컴포넌트들을 구성
2. Tailwind CSS를 이용해서 공통의 레이아웃을 구성하고 이를 통해서 페이지 기반의 애플리케이션을 구성

Index

2.1 개발 목표의 이해

2.2 React-Router 설정

2.3 <Link>를 통한 이동

2.4 레이아웃 컴포넌트와 children

2.5 상단 메뉴 컴포넌트 구성

2.6 하위 경로의 설정과 <Outlet>

2.7 todo/list 경로의 처리

2.8 중첩 라우팅의 분리와 리다이렉션(Redirection)

2.9 URL Params 사용하기

2.10 경로 처리를 위한 useParams( )

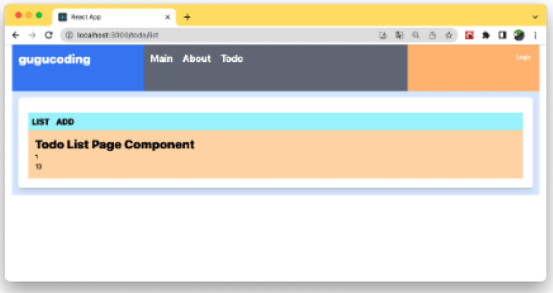
2.11동적 페이지 이동

1. 개발 목표의 이해

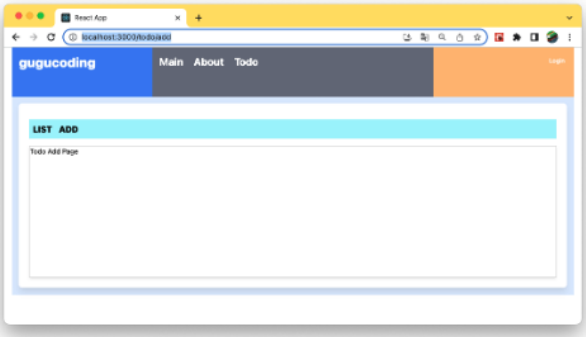
애플리케이션의 구성

→ 목록 페이지, 등록 페이지, 조회 페이지, 수정/삭제 페이지

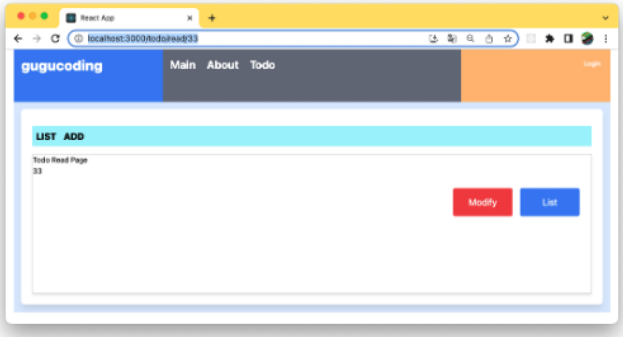
http://localhost:3000/todo/list



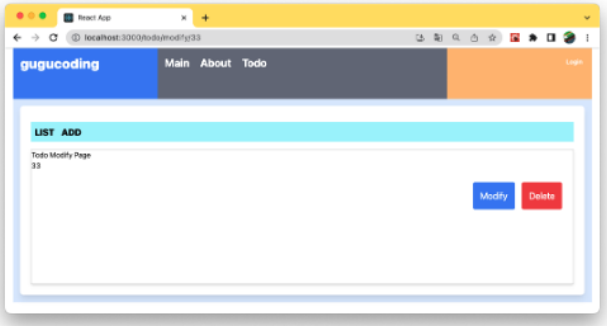
http://localhost:3000/todo/add



http://localhost:3000/todo/read/33



http://localhost:3000/todo/modify/33



React-Router 추가

→ React-Router 모듈을 추가

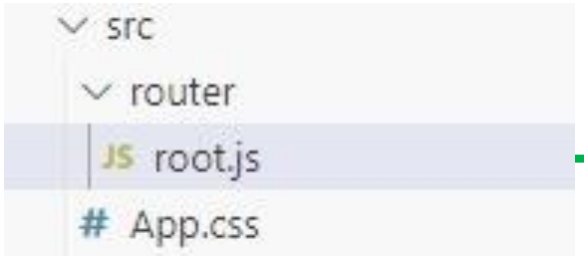
npm install react-router-dom

```
{
  "name": "mall",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.11.2",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
```

2. React-Router 설정

기본 라우팅 설정

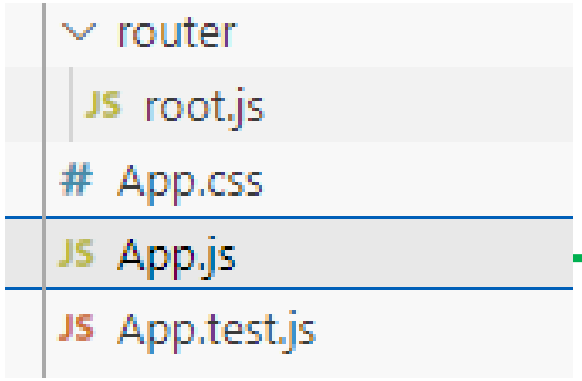
→ root.js 파일 : 기본 라우팅 설정 추가




```
const { createBrowserRouter } = require("react-router-dom");
const root = createBrowserRouter([
])
export default root;
```

기본 라우팅 설정

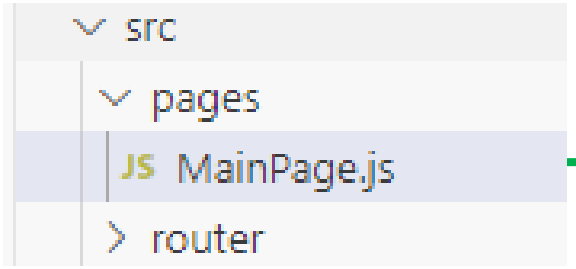
→ App.js 파일 수정




```
import {RouterProvider} from "react-router-dom";
import root from "./router/root";

function App() {
  return (
    <RouterProvider router={root}></RouterProvider>
  );
}

export default App;
```

페이지용 컴포넌트 추가와 설정

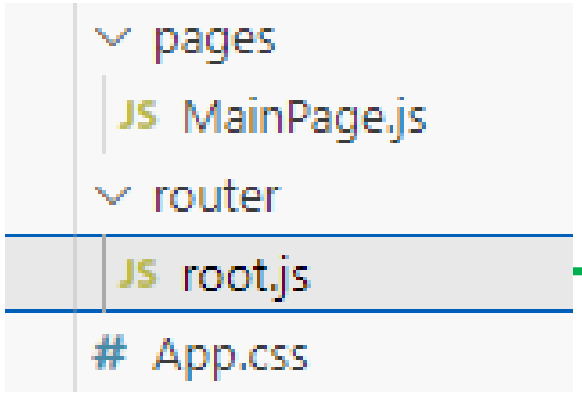
→ pages라는 이름의 폴더 생성, **MainPage.js 파일 추가**




```jsx
const MainPage = () => {
  return (
    <div className=" text-3xl">
      <div>Main Page</div>
    </div>
  );
}

export default MainPage;
```

페이지용 컴포넌트 추가와 설정

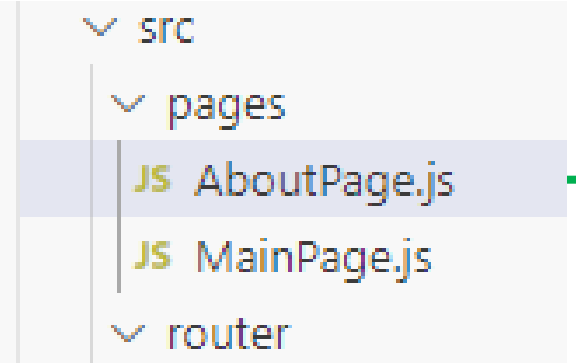
→ 첫 화면은 MainPage 컴포넌트가 보일 수 있도록 **root.js 파일에 설정**




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

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

const Loading = <div>Loading....</div>
const Main = lazy(() => import("../pages/MainPage"))

const root = createBrowserRouter([

  {
    path: "",
    element: <Suspense fallback={Loading}><Main/></Suspense>
  }

])

export default root;
```

페이지용 컴포넌트 추가와 설정 : lazy( ) 지연로딩 동작 확인

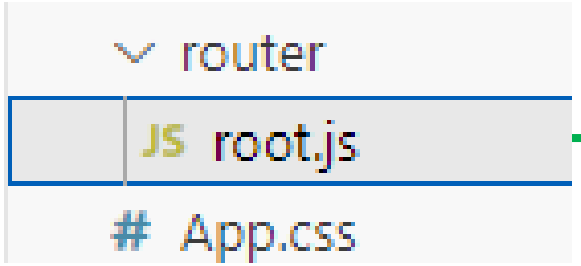
→ 컴포넌트로 AboutPage.js 추가




```
const AboutPage = () => {
  return (
    <div className=" text-3xl">About Page</div>
   );
}

export default AboutPage;
```

페이지용 컴포넌트 추가와 설정 : lazy( ) 지연로딩 동작 확인

→ '/about' 경로에서 페이지가 보이도록 root.js 설정




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

const { createBrowserRouter } = require("react-router-dom");
const Loading = <div>Loading....</div>
const Main = lazy(() => import("../pages/MainPage"))

const About = lazy(() => import("../pages/AboutPage"))

const root = createBrowserRouter([

  {
    path: "",
    element: <Suspense fallback={Loading}><Main/></Suspense>
  },
  {
    path: "about",
    element: <Suspense fallback={Loading}><About/></Suspense>
  }
])

export default root;
```

3. <Link>를 통한 이동

<Link>를 통한 이동

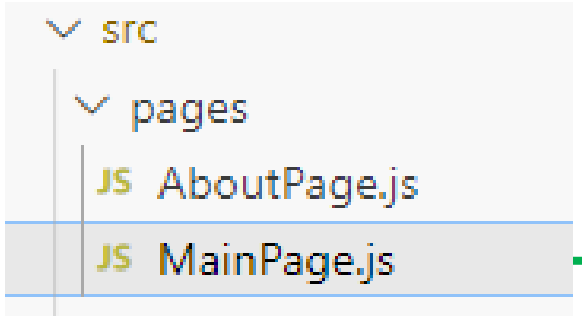
→ 리액트는 SPA로, 브라우저 주소창을 통해 컴포넌트 출력 가능.

→ 주소창 변경은 애플리케이션 전체 로딩과 처리를 의미.

→ SPA에서는 새 창 열거나 '새로고침' 주의해야 함.

→ React-Router에서 <a> 태그 사용을 피해야 함.

React-Router를 이용하는 경우에 다른 경로에 대한 링크는 <Link>를 이용

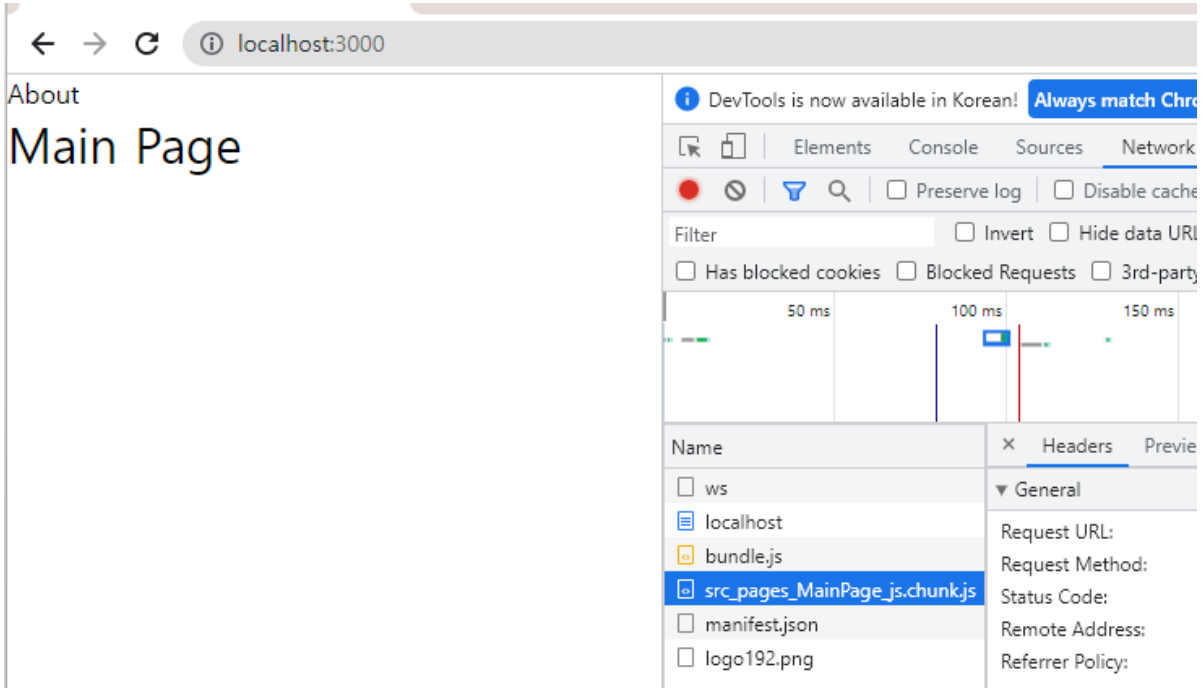



```jsx
import { Link } from "react-router-dom";

const MainPage = () => {
  return (
    <div>
      <div className="flex">
        <Link to={'/about'}>About</Link>
      </div>
      <div className=" text-3xl">Main Page</div>
    </div>

  );
}

export default MainPage;
```

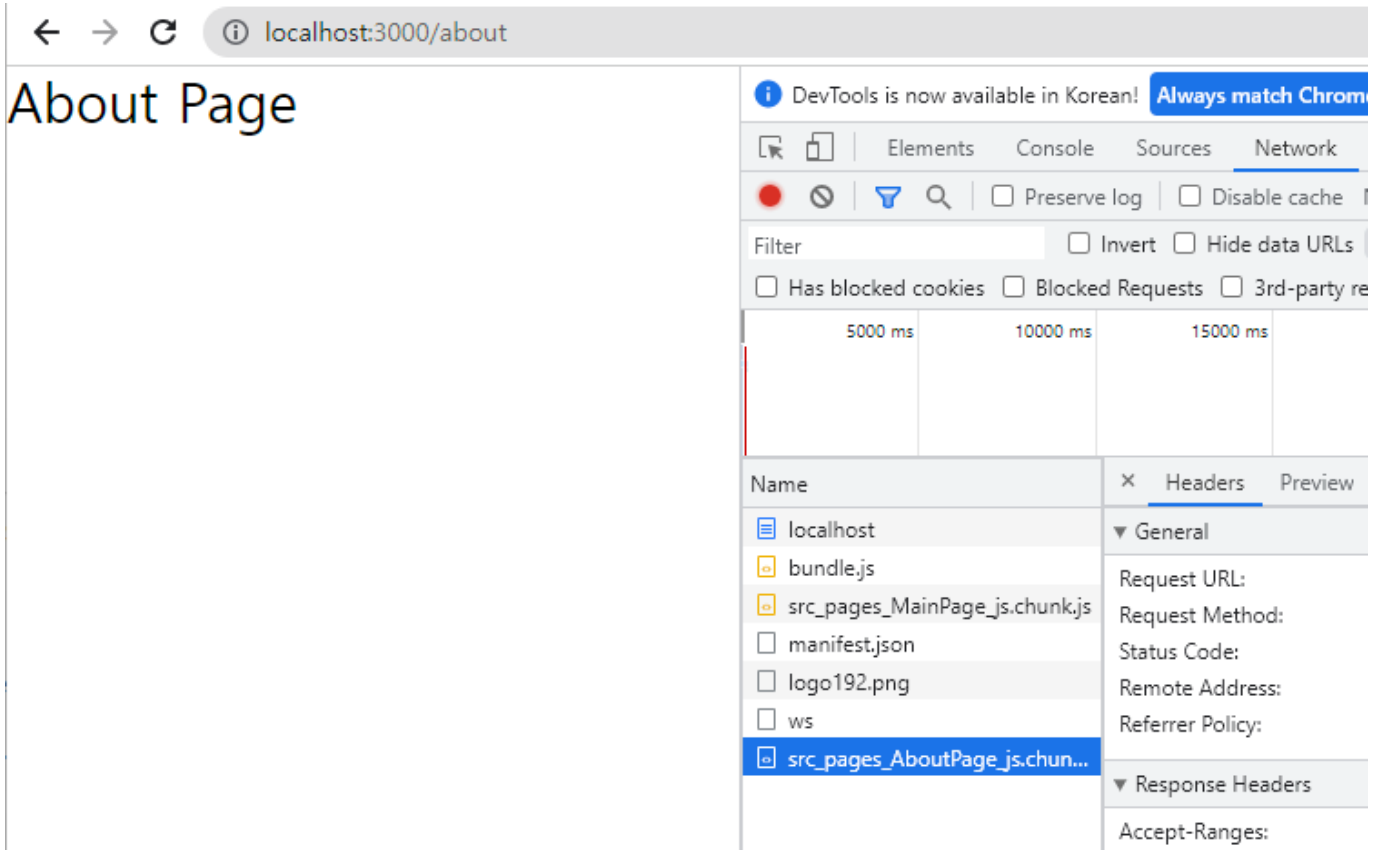
<Link>는 리액트 내부에서 컴포넌트 처리.

→ <Suspense>와 lazy( )로 필요 시점에 컴포넌트 로딩

→ '/' 경로 접근 시 MainPage만 로딩하여 보여줌.



'About'을 클릭하면 AboutPage 컴포넌트만 추가적으로 로딩되는 것을 확인



SPA 리액트 애플리케이션의 단점

→초기 실행 시간이 오래 걸림.

해결 방법 : 분할 로딩

→ <Suspense>와 <Lazy>를 이용한 코드 분할(Code Splitting) 적용.

페이지 컴포넌트들의 레이아웃

→ React-Router를 이용하여 웹 페이지 간 이동처럼 컴포넌트 처리.

→ <Link> 등으로 브라우저 새로 고침 최소화, 페이지 컴포넌트 간 이동 용이.

→ 공통 레이아웃 템플릿 구성, 메뉴 구조로 재사용 가능한 링크 처리.

4. 레이아웃 컴포넌트와 children

layouts 폴더를 생성, BasicLayout.js 컴포넌트를 생성

→ BasicLayout 컴포넌트

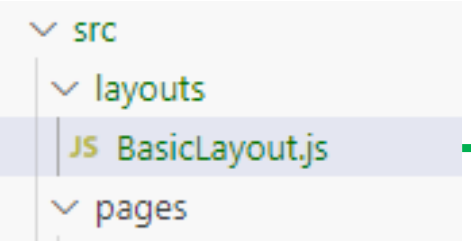
상단에 공통 메뉴와 링크, 아래로 페이지 컴포넌트 출력.

→ 'children' 속성으로 컴포넌트 내부에 다른 컴포넌트 적용 가능.

→ 간단한 구조

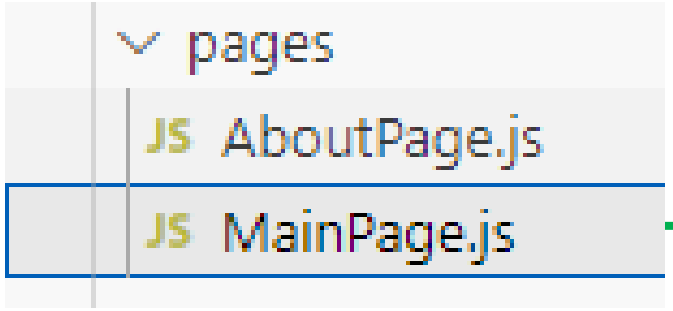
<nav>와 로 구분하여 작성.

layouts 폴더를 생성, BasicLayout.js 컴포넌트를 생성



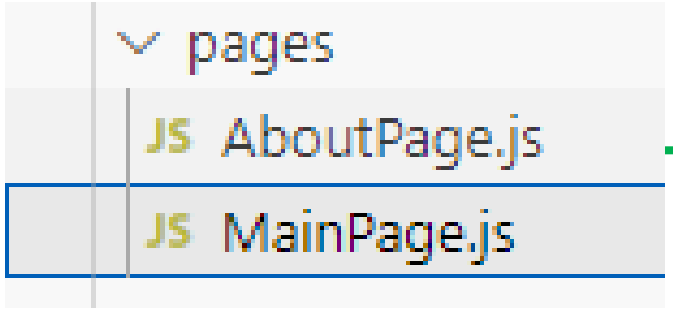

```
const BasicLayout = ({children}) => {
 return (
 <>
  <header className="bg-teal-400 p-5">
   <h1 className="text-2xl md:text-4xl">Header</h1>
  </header>
  <div className="bg-white my-5 w-full flex flex-col space-y-4 md:flex-row md:space-x-4
md:space-y-0">
   <main className="bg-sky-300 md:w-2/3 lg:w-3/4 px-5 py-40">
     {children}
   </main>
   <aside className="bg-green-300 md:w-1/3 lg:w-1/4 px-5 py-40">
    <h1 className="text-2xl md:text-4xl"> Sidebar </h1>
   </aside>
  </div>
 </>
 );
}

export default BasicLayout;
```

MainPage 컴포넌트의 내용은 <BasicLayout>을 이용해서 수정




```jsx
import BasicLayout from "../layouts/BasicLayout";

const MainPage = () => {
  return (
    <BasicLayout>

      <div className=" text-3xl">Main Page</div>
    </BasicLayout>

   );
}

export default MainPage;
```

AboutPage : <BasicLayout> 을 적용

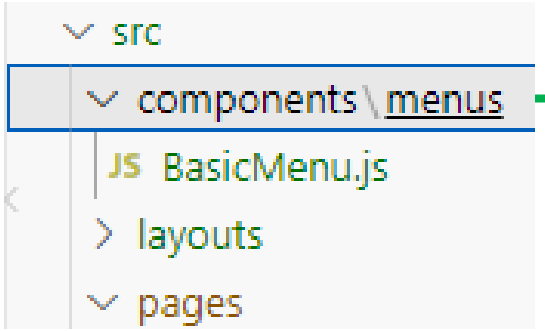



```jsx
import BasicLayout from "../layouts/BasicLayout";

const AboutPage = () => {
  return (
    <BasicLayout>
      <div className=" text-3xl">About Page</div>
    </BasicLayout>

  );
}

export default AboutPage;
```

5. 상단 메뉴 컴포넌트 구성

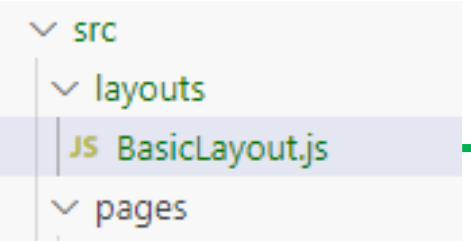
레이아웃 화면에 메뉴 부분 : 별도의 컴포넌트 활용.

→ layouts 폴더의 menu 폴더에 **BasicMenu.js** 파일 추가.




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

const BasicMenu = () => {
 return (
 <nav id='navbar' className=" flex bg-blue-300">
  <div className="w-4/5 bg-gray-500" >
   <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>
   </ul>
  </div>
  <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;
```

BasicLayout에 상단 메뉴로 추가 → 각 화면에 공통의 메뉴가 출력 되도록 구성




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

const BasicLayout = ({children}) => {
 return (
 <>
  <BasicMenu></BasicMenu>
  <div className="bg-white my-5 w-full flex flex-col space-y-4 md:flex-row md:space-x-4
md:space-y-0">
   <main className="bg-sky-300 md:w-2/3 lg:w-3/4 px-5 py-40">
     <h1 className="text-2xl md:text-4xl">{children}</h1>
   </main>
   <aside className="bg-green-300 md:w-1/3 lg:w-1/4 px-5 py-40">
     <h1 className="text-2xl md:text-4xl">Sidebar</h1>
   </aside>
  </div>
 </>
 );
}

export default BasicLayout;
```

새로운 단위 기능과 라우팅

→ 애플리케이션 컴포넌트가 증가하면 React-Router 설정과 메뉴 구조 복잡해짐.

→ 기능들을 묶어 '모듈'로 구성(예: 게시판, 회원, 상품).

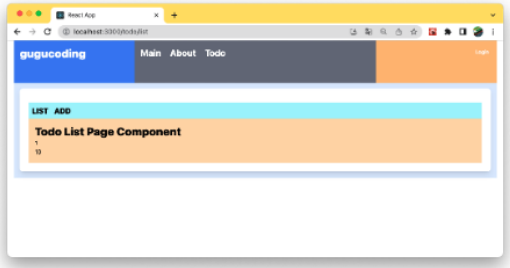
→ 각 모듈은 내부적으로 자체 메뉴를 가짐.

→ React-Router의 <Outlet>을 활용하여 모듈의 경로 구성.

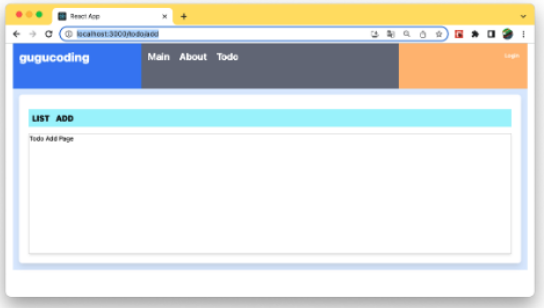
새로운 단위 기능과 라우팅

/todo/list	목록 페이지
/todo/add	등록 페이지
/todo/read/33	조회 페이지 (with 번호)
/todo/modify/33	수정/삭제 페이지 (with 번호)

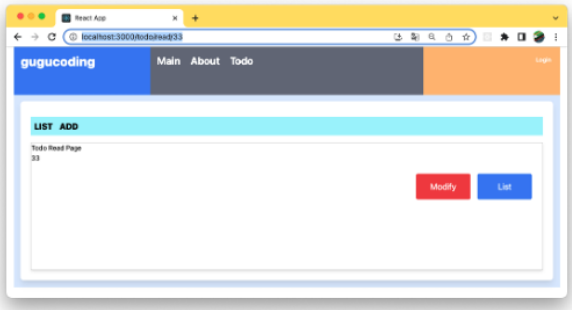
http://localhost:3000/todo/list



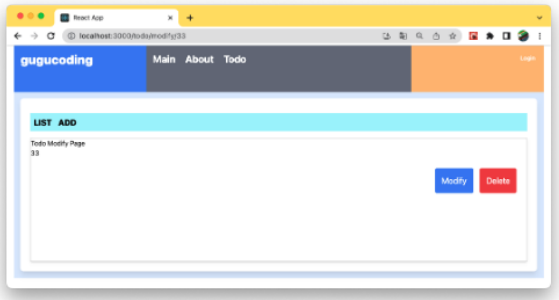
http://localhost:3000/todo/add



http://localhost:3000/todo/read/33

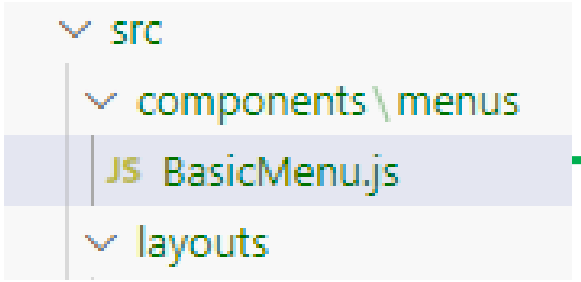


http://localhost:3000/todo/modify/33



Todo

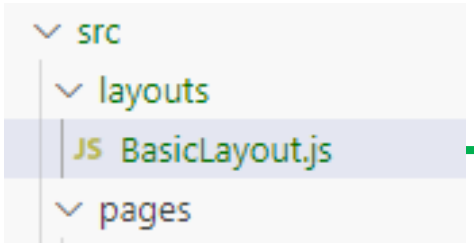
→ Todo 기능 위해 BasicMenu 컴포넌트에 '/todo/' 링크 추가




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

const BasicMenu = () => {
 return (
 <nav id='navbar' className=" flex bg-blue-300">
  <div className="w-4/5 bg-gray-500" >
   <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>
    <li className="pr-6 text-2xl"><Link to={'/todo/'}>Todo</Link></li>
   </ul>
  </div>
  <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;
```

6. 하위 경로의 설정과 <Outlet>

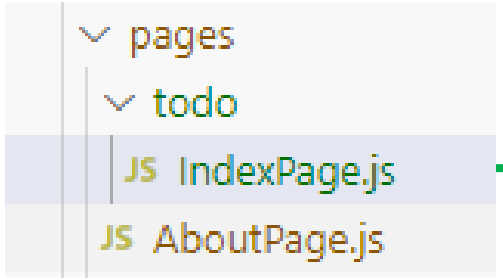
BasicLayout에서 padding, flex 관련 설정을 조정




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

const BasicLayout = ({children}) => {
 return (
 <>
  {/* 기존 헤더 대신 BasicMenu*/ }
  <BasicMenu/>
  {/* 상단 여백 my-5 제거 */}
  <div className="bg-white my-5 w-full flex flex-col space-y-1 md:flex-row md:space-x-1 md:space-y-
0">
   {/* 상단 여백 py-40 변경 flex 제거 */}
   <main className="bg-sky-300 md:w-2/3 lg:w-3/4 px-5 py-5">{children}</main>
   {/* 상단 여백 py-40 제거 flex 제거 */}
   <aside className="bg-green-300 md:w-1/3 lg:w-1/4 px-5 flex py-5">
    <h1 className="text-2xl md:text-4xl">Sidebar</h1>
   </aside>
  </div>
 </>
 );
}

export default BasicLayout;
```

pages 폴더에 todo 폴더 생성, IndexPage.js 추가.

→ IndexPage.js에서 BasicLayout을 이용해 Todo와 관련된 메뉴 및 화면 구성.



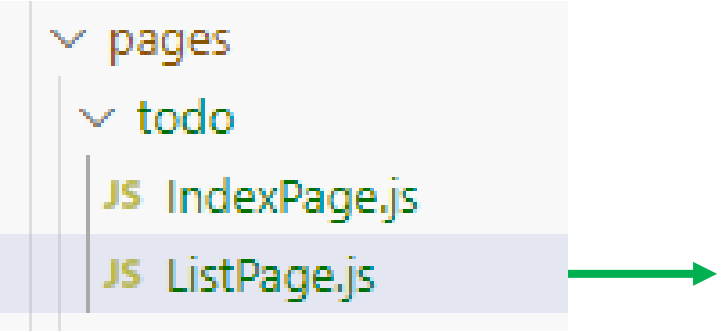

```
import { Outlet } from "react-router-dom";
import BasicLayout from "../../layouts/BasicLayout";

const IndexPage = () => {

 return (
  <BasicLayout>
   <div className="w-full flex m-2 p-2 ">
    <div className="text-xl m-1 p-2 w-20 font-extrabold text-center underline">LIST</div>
    <div className="text-xl m-1 p-2 w-20 font-extrabold text-center underline">ADD</div>
   </div>
   <div className="flex flex-wrap w-full">
    <Outlet/>
   </div>
  </BasicLayout>
  );
}

export default IndexPage;
```

라우팅 설정위해 router/root.js에 '/todo' 관련 경로를 추가


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

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

...

const TodoIndex = lazy(() => import("../pages/todo/IndexPage"))

const root = createBrowserRouter([
  ....
  {
    path: "todo",
    element: <Suspense fallback={Loading}><TodoIndex/></Suspense>
  }
])

export default root;
```

7. todo/list 경로의 처리

pages/todo 폴더에 ListPage.js 파일을 추가




```
return (
    <div className="p-4 w-full bg-orange-200 ">
      <div className="text-3xl font-extrabold">
        Todo List Page Component
      </div>
    </div>
  );
}

export default ListPage;
```

React-Router의 중첩 라우팅


```
...생략
const TodoList =  lazy(() => import("../pages/todo/ListPage"))

const root = createBrowserRouter([
  ...생략
  {
    path: "todo",
    element: <Suspense fallback={Loading}><TodoIndex/></Suspense>,
    children: [
      {
        path: "list",
        element: <Suspense fallback={Loading}> <TodoList/> </Suspense>
      }
    ]
  }
])
export default root;
```

8. 중첩 라우팅의 분리와 리다이렉션(Redirection)

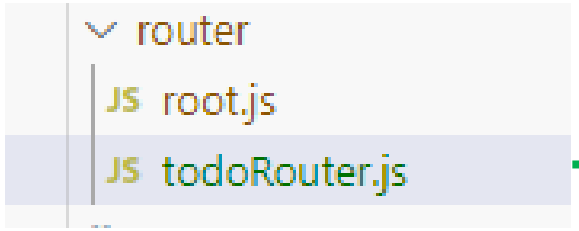
중첩 라우팅의 분리

→ 하나의 라우팅 설정에서 children 속성으로 중첩 라우팅 설정 가능.

→ 페이지가 많아지면 root.js 파일 복잡해짐.

→ 별도의 함수에서 children 속성값에 해당하는 설정 반환 방식으로 가독성 향상.

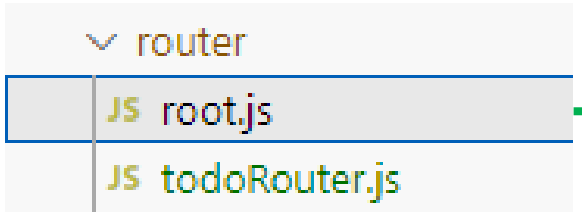
중첩 라우팅의 분리




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

const Loading = <div>Loading....</div>
const TodoList = lazy(() => import("../pages/todo/ListPage"))

const todoRouter = () => {

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

}

export default todoRouter;
```

중첩 라우팅의 분리

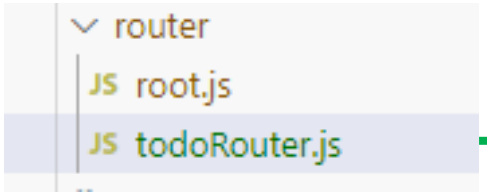
→ root.js의 라우팅 설정 수정




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

const { createBrowserRouter } = require("react-router-dom");
const Loading = <div>Loading....</div>
const Main = lazy(() => import("../pages/MainPage"))
const About = lazy(() => import("../pages/AboutPage"))
const TodoIndex = lazy(() => import("../pages/todo/IndexPage"))
const root = createBrowserRouter([
 {
  path: "", element: <Suspense fallback={Loading}><Main/></Suspense>
 },
 {
  path: "about", element: <Suspense fallback={Loading}><About/></Suspense>
 },
 {
  path: "todo", element: <Suspense fallback={Loading}><TodoIndex/></Suspense>,
  children: todoRouter()
 }
])

export default root;
```

리다이렉션 처리

→ <Navigate>의 replace 속성을 이용해서 리다이렉션을 처리




```jsx
import { Suspense, lazy } from "react";
import { Navigate } from "react-router-dom";

const Loading = <div>Loading....</div>
const TodoList = lazy(() => import("../pages/todo/ListPage"))

const todoRouter = () => {
 return [
  {
   path: "list", element: <Suspense fallback={Loading}><TodoList/></Suspense>
  },
  {
   path: "", element: <Navigate replace to="list"/>
  }
 ]
}

export default todoRouter;
```

9. URL Params 사용하기

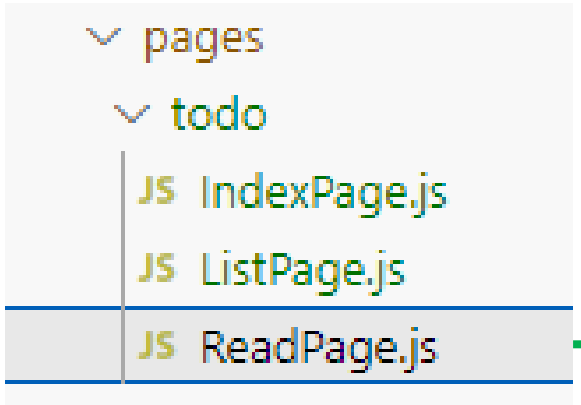
URL Params

→ Todo 목록 페이지에서 조회 페이지로 이동 시 경로 변경.

예: '/todo/read/33'와 같이 데이터가 있는 경로.

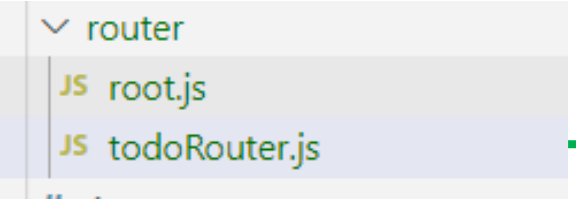
→ React-Router에서 **':'를 활용**하여 데이터 전달.

ReadPage.js 파일 추가




```jsx
const ReadPage = () => {

 return (
  <div className="text-3xl font-extrabold">
   Todo Read Page Component
  </div>
 );

}

export default ReadPage;
```

router/todoRouter.js에 ReadPage컴포넌트 경로 설정




```
import { Suspense, lazy } from "react";
import { Navigate } from "react-router-dom";

const todoRouter = () => {
 const Loading = <div>Loading....</div>
 const TodoList = lazy(() => import("../pages/todo/ListPage"))
 const TodoRead = lazy(() => import("../pages/todo/ReadPage"))
 return [
  {
   path: "list", element: <Suspense fallback={Loading}><TodoList/></Suspense>
  },
  {
   path: "", element: <Navigate replace to="/todo/list"/>
  },
  {
   path: "read/:tno", element: <Suspense fallback={Loading}><TodoRead/></Suspense>
  },
 ]
}
export default todoRouter;
```

10. 경로 처리를 위한 useParams( )

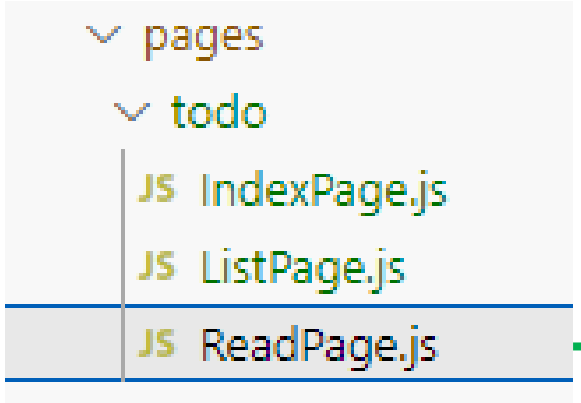
경로 처리를 위한 useParams( )

→ 특정 번호의 경로 사용 시, 컴포넌트에서 주소창의 일부 활용 필요.

→ React-Router의 useParams( )로 지정된 변수 추출 가능.

경로 처리를 위한 useParams( )

ReadPage 컴포넌트에서 tno 변수로 전달된 값 출력.



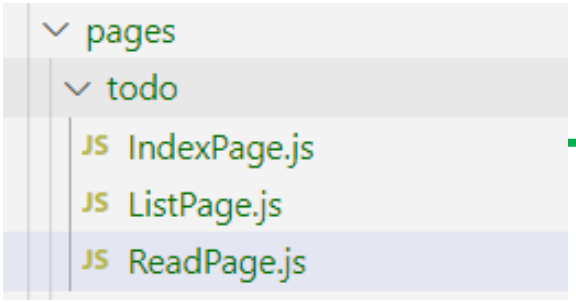

```jsx
import { useParams } from "react-router-dom";

const ReadPage = () => {

 const {tno} = useParams()

 return (
  <div className="text-3xl font-extrabold">
   Todo Read Page Component {tno}
  </div>
 );

}

export default ReadPage;
```

경로 처리를 위한 useParams( )

쿼리스트링은 useSearchParams( )를 이용


```jsx
import { useSearchParams } from "react-router-dom";

const ListPage = () => {

 const [queryParams] = useSearchParams()

 const page = queryParams.get("page") ? parseInt(queryParams.get("page")) : 1
 const size = queryParams.get("size") ? parseInt(queryParams.get("size")) : 10

 return (
 <div className="p-4 w-full bg-white">
  <div className="text-3xl font-extrabold">
   Todo List Page Component {page} --- {size}
  </div>
 </div>
  );
}

export default ListPage;
```

쿼리스트링은 useSearchParams( )를 이용




```jsx
import { useSearchParams } from "react-router-dom";

const ListPage = () => {

  const [queryParams] = useSearchParams()
  const page = queryParams.get("page") ||1
  const size = queryParams.get("size") ||10

  return (
    <div className="p-4 w-full bg-orange-200 ">
      <div className="text-3xl font-extrabold">
        Todo List Page Component
      </div>
      <div>{page}</div>
      <div>{size}</div>
    </div>
   );
}

export default ListPage;
```

useNavigate( )

→ React-Router를 사용할 때, 링크로 이동하는 경우도 있지만 동적 데이터 처리로 이동하는 경우가 더 많음.

→ useNavigate( )를 활용하여 프로그램을 통해 데이터 동적 이동 처리.

경로 처리를 위한 useParams( )

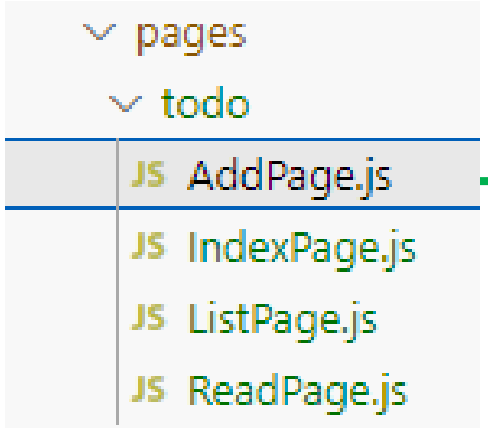
useNavigate( )

→ IndexPage 컴포넌트의 각 링크에 대한 이벤트 처리


```
import { Outlet, useNavigate } from "react-router-dom";
import BasicLayout from "../../layouts/BasicLayout";
import { useCallback } from "react";

const IndexPage = () => {

 const navigate = useNavigate()

 const handleClickList = useCallback(() => { navigate({ pathname:'list' }) })
 const handleClickAdd = useCallback(() => { navigate({ pathname:'add' }) })

 return (
  <BasicLayout>
   <div className="w-full flex m-2 p-2 ">
    <div className="text-xl m-1 p-2 w-20 font-extrabold text-center underline"
         onClick={handleClickList}> LIST </div>
    <div className="text-xl m-1 p-2 w-20 font-extrabold text-center underline"
         onClick={handleClickAdd}> ADD </div>
   </div>
   <div className="flex flex-wrap w-full"> <Outlet/>  </div>
  </BasicLayout>
  );
}

export default IndexPage;
```

useNavigate( )

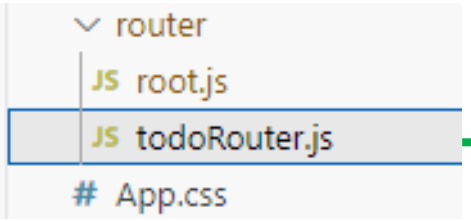
→ AddPage.js 파일을 생성




```jsx
const AddPage = () => {


 return (
  <div className="text-3xl font-extrabold">
   Todo Add Page
  </div>
 );

}

export default AddPage;
```

useNavigate( )

→ router/todoRouter.js에는 '/todo/add' 경로에 대한 설정을 추가




```
import { Suspense, lazy } from "react";
import { Navigate } from "react-router-dom";

const Loading = <div>Loading....</div>
const TodoList = lazy(() => import("../pages/todo/ListPage"))
const TodoRead = lazy(() => import("../pages/todo/ReadPage"))
const TodoAdd = lazy(() => import("../pages/todo/AddPage"))

const todoRouter = () => {
 return [
  { path: "list", element: <Suspense fallback={Loading}><TodoList/></Suspense> },
  { path: "", element: <Navigate replace to="list"/> },
  { path: "read/:tno", element: <Suspense fallback={Loading}><TodoRead/></Suspense> },
  { path: "add", element: <Suspense fallback={Loading}><TodoAdd/></Suspense> }
 ]
}

export default todoRouter;
```

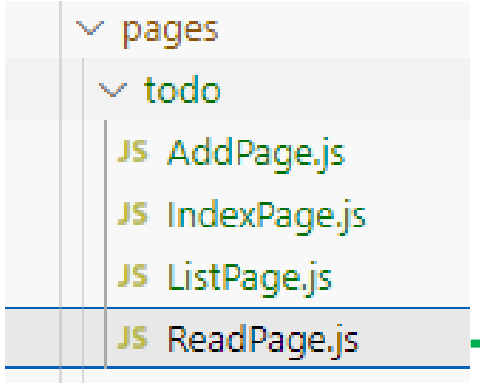
11. 동적 페이지 이동

동적 페이지 이동

→ 수정/삭제 작업은 매번 다른 번호(tno)를 사용

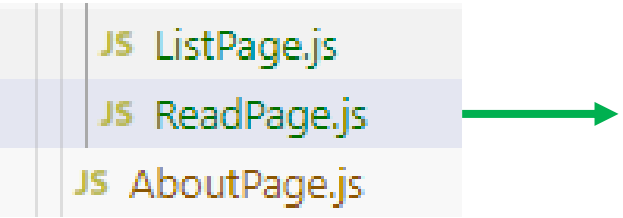
→ useParams( )로 찾은 번호를 이용하여 동적 처리

동적 페이지 이동 : 조회 → 수정/삭제




```jsx
import { useCallback } from "react";
import { useNavigate, useParams } from "react-router-dom";

const ReadPage = () => {

 const {tno} = useParams()
 const navigate = useNavigate()
 const moveToModify = useCallback((tno) => {
  navigate({pathname:`/todo/modify/${tno}`})
 },[tno])

 return (
  <div className="text-3xl font-extrabold"> Todo Read Page Component {tno}
   <div> <button onClick={() => moveToModify(33)}>Test Modify</button> </div>
  </div>
 );

}

export default ReadPage;
```

동적 페이지 이동 : 조회 → 수정/삭제



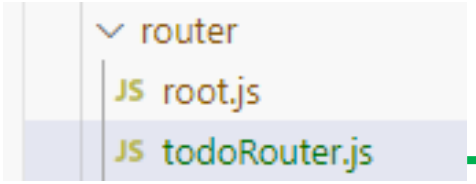

```jsx
import { useCallback } from "react";
import { createSearchParams, useNavigate, useParams, useSearchParams } from "react-router-dom";

const ReadPage = () => {
 const {tno} = useParams()
 const navigate = useNavigate()

 const [queryParams] = useSearchParams()

 const page = queryParams.get("page") ? parseInt(queryParams.get("page")) : 1
 const size = queryParams.get("size") ? parseInt(queryParams.get("size")) : 10
 const queryStr = createSearchParams({page,size}).toString()
 const moveToModify = useCallback((tno) => {
  navigate({ pathname: `/todo/modify/${tno}`, search: queryStr })
 },[tno, page, size])

 return (
  <div className="text-3xl font-extrabold">
   Todo Read Page Component {tno}
   <div> <button onClick={() => moveToModify(tno)}>Test Modify</button> </div>
  </div>
 );
}

export default ReadPage;
```

동적 페이지 이동 : 수정/삭제 페이지


```jsx
const ModifyPage = ({tno}) => {

 return (
  <div className="text-3xl font-extrabold">
   Todo Modify Page
  </div>
  );

}

export default ModifyPage;
```

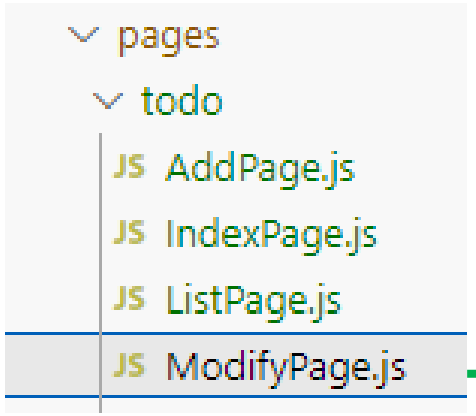
동적 페이지 이동 : 수정/삭제 페이지

→ 라우팅 설정




```
import { Suspense, lazy } from "react";
import { Navigate } from "react-router-dom";

const Loading = <div>Loading....</div>
const TodoList = lazy(() => import("../pages/todo/ListPage"))
const TodoRead = lazy(() => import("../pages/todo/ReadPage"))
const TodoAdd = lazy(() => import("../pages/todo/AddPage"))
const TodoModify = lazy(() => import("../pages/todo/ModifyPage"))

const todoRouter = () => {

 return [
  …생략
  { path: "modify/:tno", element: <Suspense fallback={Loading}><TodoModify/></Suspense> }
 ]
}

export default todoRouter;
```

동적 페이지 이동 : 수정/삭제 처리 후 이동




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

const ModifyPage = ({tno}) => {

 const navigate = useNavigate()

 const moveToRead = () => {
  navigate({pathname:`/todo/read/${tno}`})
 }

 const moveToList = () => {
  navigate({pathname:`/todo/list`})
 }

 return (
  <div className="text-3xl font-extrabold"> Todo Modify Page </div>
  );
}

export default ModifyPage;
```

감사합니다.