



IBK클라우드 시스템 구축

화면 개발 가이드

Version 1.2

2025년 5월 7일

IBK 클라우드 포탈 시스템 구축

클라우드팀

- 목 차 -

1. 개요	4
2. Front-End 아키텍처	5
3. 명명 표준	15
4. 퍼블리싱	20
5. 개발	25
6. 예제 (게시판 생성, 조회, 수정)	68
7. 배포	89
8. 기타	89
9. 별첨	91

1. 개요

본 클라우드 개발가이드는 IT시스템 구축 시 UI(화면) 개발에 필요한 표준 및 기술 사항에 대한 내용을 담고 있다.

주요 내용으로는 화면 디자인 표준사항, 화면 컨트롤의 주요 사용 방법, 데이터 포맷에 관한 사항, 서버에 요청을 보내는 방법 등 화면 개발에 필요한 전반적인 내용 및 본 프로젝트만의 특이 사항 등을 설명한다.

2. Front-End 아키텍처

2.1. Design Token

2.1.1. 개념

디자인 토큰은 기업의 다양한 Web/App 서비스 등의 일관적인 브랜딩을 유지하기 위해 대표적인 글로벌 CRM SaaS 서비스 기업인 Salesforce에서 고안한 방법으로 Adobe가 Detail한 영역들을 발전시킨 체계적인 컴포넌트, 모듈, 화면에 대한 방법론이다.

국내 테크니컬 기업인 토스, 카카오, 네이버 등 일관된 브랜딩을 제공하는 곳에서 디자인 시스템으로 체계적으로 정리해둔 Document를 활용하여 브랜딩 운용, 유지보수 등 다양한 측면에서 일관되게 사용되고 있다.

2.1.2. 형태

디자인 토큰 파일은 scss, json, xml 등 다양한 포맷으로 만들어질 수 있으며 토큰에는 디자인의 기본 요소 (색상값, 폰트, 간격, 여백 등)에 대한 내용이 들어있다.

2.1.3. 도입 이유

전통적인 방식에서는 가이드 문서를 보고 수동으로 디자인 통일성을 유지하였으므로 디자인을 유지보수하고 개선할 때 작업 비용이 많이 들어갔다.

이에 비해 디자인 토큰을 이용하여 화면을 만들면 디자인이 변경되었을 때, 퍼블리셔 작업 등 중간 과정 없이 토큰 파일만 바꾸면 전체 시스템의 디자인을 일괄적으로 손쉽게 바꿀 수 있어 유지보수 비용을 줄일 수 있다.

2.1.4. 제작 방법

Figma 툴로 디자인을 하고, 피그마의 Figma Tokens Plugin을 사용하여 디자인 토큰을 뽑아내어 CSS로 렌더링한 npm library를 발행한다. 이후 Framework에서 해당 라이브러리를 import 하여 그에 의존적으로 CSS스타일을 작성하여 디자인 일관성을 유지한다.

최종적으로는 CSS 형태이므로 이 디자인 토큰을 사용하는 Front-End Framework 는 어떤 종류여도 상관없다. (Vue, Angular, React, Svelte 등)

Design Token CSS Library	프로젝트에서 사용시 구조
<pre>:root { --primary-color: blue --secondary-color: red }</pre>	
Framework > .css	
<pre>button { color: var(--primary-color); }</pre>	

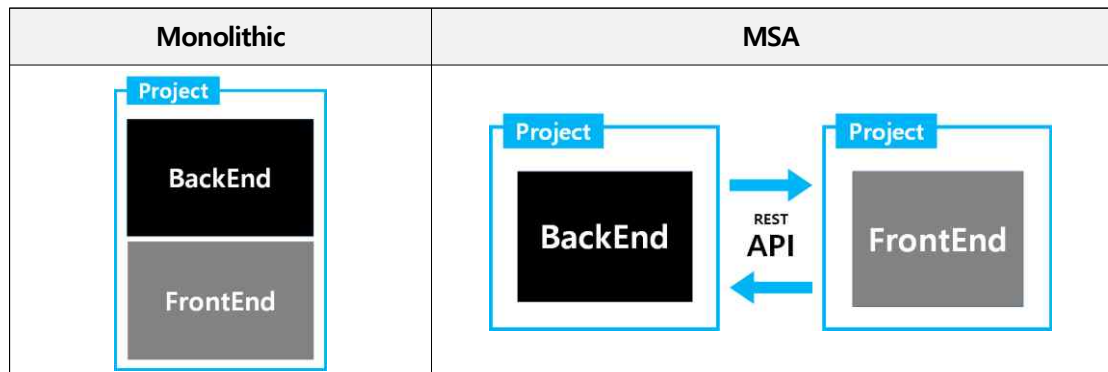
* 디자인이 다른 경우 새로운 디자인 토큰을 발행하여 사용

* Figma 는 Web based 디자인 툴이므로 인터넷 망에서 작업이 필요

2.2. MSA

모놀리식 아키텍처에서는 하나의 프로젝트에서 Back-End(서버, DB 영역)와 Front-End(화면 영역)을 한꺼번에 관리하는 경우가 많아, Front-End 단을 변경하면 Back-End 에도 영향이 있을 수밖에 없는 구조였다.

이번에 MSA 를 도입하며 Back-End 영역과 Front-End 영역을 분리하여 api 로만 통신하게 함으로서 Front-End 영역을 독립적으로 관리하여 보다 안전하고 효율적으로 유지보수를 할 수 있도록 한다.

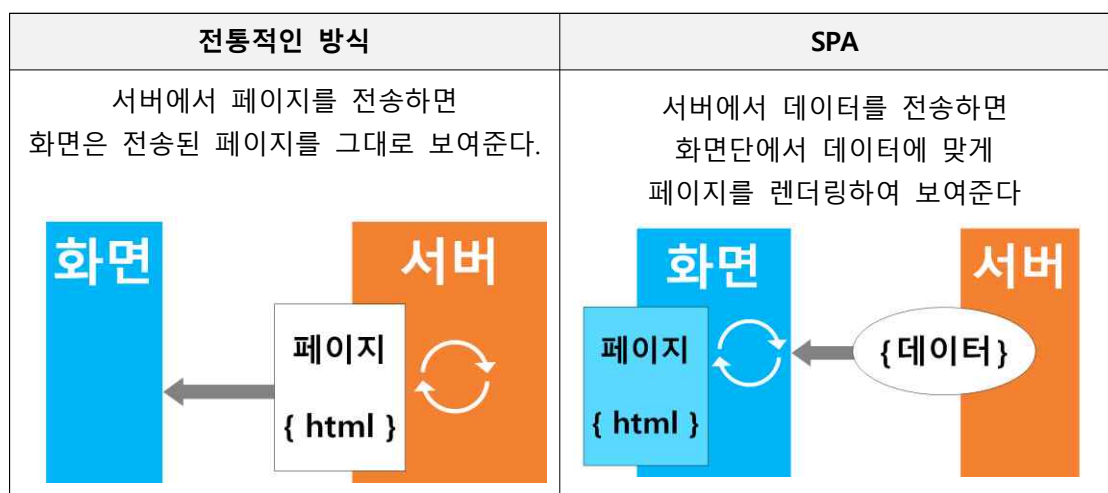


2.3. SPA (Single Page Application)

전통적인 웹 사이트에서는 문서 하나에 전달되는 파일의 용량이 적었다. 그래서 어떤 요소를 한번 클릭하면 서버에서 새로운 페이지를 통째로 렌더링하여 전송해 주었다.

그러나 현대에 이르러 웹 사이트가 고도화됨에 따라 한 페이지에 해당하는 페이지 용량이 커져갔고 매번 새로운 페이지를 전달하는 일이 번거로워지게 되었다.

이러한 문제를 해결하기 위해 등장한 것이 SPA(Single Page Application)이다. 이 방식에서는 웹 사이트의 모든 리소스를 한번 내려 받은 후, 화면을 새로고침 하지 않고 서버와는 데이터만 주고받으며 Front-End Framework 단에서 데이터를 기반으로 화면을 동적으로 렌더링하여 출력한다.

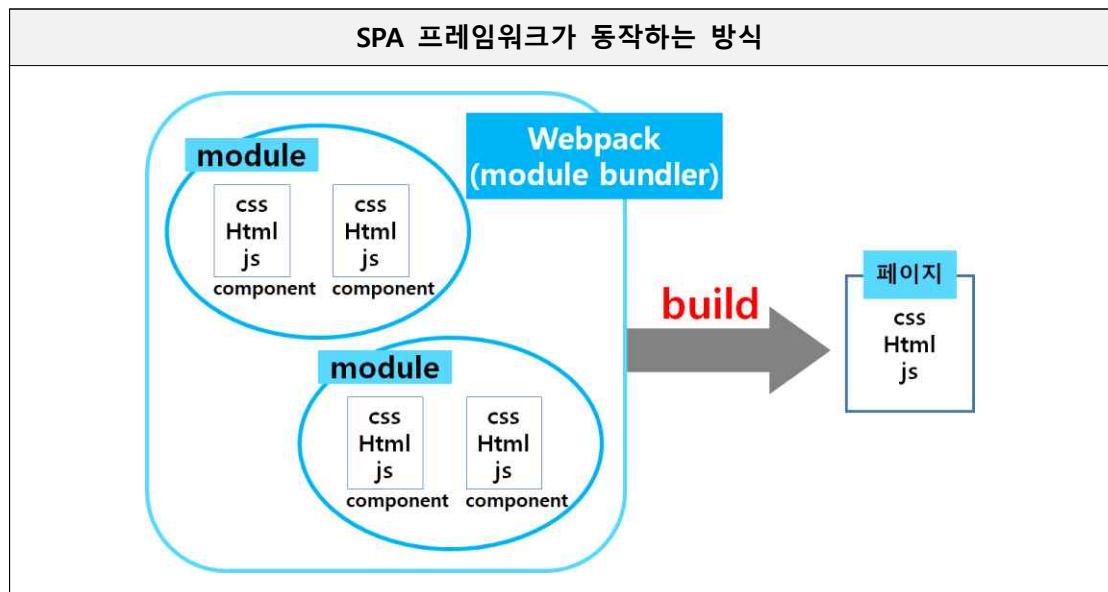


2.4. Front-End Framework

2.4.1. Front-End Framework 소개

화면을 SPA 로 만들게 되면 기존 방식보다 Front-End 단에서 복잡한 처리를 해야하기 때문에 이를 위한 개발 편의성을 제공하는 다양한 라이브러리와 프레임워크가 시중에 출시되어 있다. 가장 많이 사용되는 SPA 프레임워크로는 React, Angular, Vue 가 있다.

이들 프레임워크의 문법은 다 다르지만 기본적으로는 비슷한 방식을 따른다. 먼저 css, html, js로 이루어진 컴포넌트 단위로 화면을 개발한다. 컴포넌트는 페이지 단위로 나눌 수도 있고 공통된 기능끼리 묶을 수도 있다. 여러 개의 컴포넌트가 모인 것을 module 이라고 부른다. 이 module들을 module bundler를 이용하여 (여러 가지가 있지만 대표적으로 webpack 이 있다) 묶고, 빌드하면 전체 모듈을 압축 및 최적화하여 하나의 페이지로 렌더링하여 화면에 보여준다.



클라우드 포탈은 React를 사용하여 만든다.

2.4.2. 클라우드 포탈 React 채택 사유

Angular, Vue 와 다르게 React 는 완성된 형태의 프레임워크가 아니라 MVC 중 V(View)만 담당하는 라이브러리이다. MC(Model, Controller)에 해당하는 부분은 다양한 오픈 소스들을 세팅하여 사용하는 프로젝트의 규모나 상황에 맞게 커스터마이징이 가능한 유연성이 있다. 또한 양방향 바인딩*을 사용하는 Angular(MVC 패턴), Vue(MVVM 패턴)와 달리 단방향 바인딩**(Flux 패턴)을 사용하여 화면 렌더링이 빠르다.

2022년 현재 가장 많이 사용되는 오픈소스로서 생태계가 활성화되어 있어 수많은 레퍼런스가 있고, 방대한 커뮤니티의 도움을 받을 수 있다는 장점이 있다.

문법은 기본적으로 vanilla javascript를 사용하므로 javascript에 익숙한 개발자에게는 어렵지 않다. JSX, 함수 컴포넌트 등을 알아야 하는데 홈페이지에 문서화가 잘 되어 있다.

* 양방향 바인딩 : View를 변경하면 데이터가 바뀌고, 데이터를 변경하면 View가 바뀐다. 이 때문에 개발이 편리하지만 규모가 커지면 속도가 느려질 수 있는 단점이 있다.

* 단방향 바인딩 : 데이터를 변경하면 View가 변경되지만, 그 반대는 허용하지 않는다.

2.5. React를 이용한 Front-End Framework 제작 방법 안내

2.5.1. 빈 폴더 생성

프로젝트명으로 빈 폴더를 생성한다.

2.5.2. nexus 사용 세팅

위에 생성한 폴더 안에 .npmrc 파일을 생성하고 nexus repository 주소를 세팅한다.
이후에는 업무망 안에서 npm 명령어를 치면 외부와 동일하게 작동된다.

```
.npmrc
registry=http://10.40.33.50:18081/repository/npm-default-group/
```

* nexus 가 안되는 환경에서는 2.5.3 부터의 과정을 전부 외부망(인터넷망)에서 실행한 뒤
프로젝트 파일을 통째로 압축해서 가져오면 된다.
이 경우 react-scripts를 필수로 설치해야 한다.

```
npm install react-scripts --save-dev
```

2.5.3. 프로젝트 생성

1. <https://nodejs.org> 를 방문하여 node를 설치한다.
node를 설치하면 npm 은 자동으로 설치된다.
2. IDE(개발툴) 설치 (이 프로젝트에서는 VSCode 사용)
3. 터미널 창을 열고 아래 명령어를 입력하면 기본 골격이 생성된다.

```
npx create-react-app [프로젝트 이름] --template typescript
```

4. [프로젝트 이름] 폴더 안에 위에 생성한 .npmrc 파일을 옮긴다.
[프로젝트 이름] 폴더를 루트 폴더로 놓고 아래 과정을 진행한다.
5. 아래 명령어를 입력하면 http://localhost:3000 에 화면이 실행된다.

```
npm start
```

6. 아래 명령어를 사용하여 프로젝트를 빌드한다.

```
npm build
```

프로젝트를 빌드하면 build 폴더가 만들어지며 전체 파일을 압축 및 최적화한 파일이 생성된다. 해당 폴더 내 파일들만 최종적으로 서버에 배포하면 된다.

2.5.4. 라이브러리 설치 및 사용

React 기본 세팅만으로는 Routing(페이지간 전환), 내부 저장소 사용이나 API 연동을 할 수 없고 추가적인 라이브러리를 설치하여 필요한 기능을 세팅해야한다.

1. npmjs.com 에 방문하여 필요한 라이브러리를 검색하여,
터미널 창에 아래와 같은 명령어로 프로젝트에 import 할 수 있다.
설치된 라이브러리명과 버전은 package.json에서 확인할 수 있다.

```
npm install [라이브러리명]
```

개발시에만 사용할 라이브러리는 -D를 붙여 설치한다.
(package.json 의 devDependencies에서 목록 확인)

```
npm install [라이브러리명] -D
```

2. 설치한 라이브러리 목록은 프로젝트의 package.json에서 확인할 수 있다.
추천하는 주요 라이브러리는 다음과 같다.

* **패키지 버전 명시 시 캐럿(^), 틸드(~)와 같은 범위 지정 사용을 지양하며, 명시적인 버전 고정(ex. 2.1.3)을 원칙으로 한다.** 이는 의도치 않은 마이너/패치 버전 업그레이드로 인한 불안정성, 빌드 실패 등의 리스크를 방지하기 위함이다.

라이브러리	사용이유	비고
@types/react-router-dom	페이지간 이동	라우팅
@mui/material	UI	IE11 브라우저를 지원하지 않아도 되는 상황에서 추천하는 UI 라이브러리 mui.com 참조
primereact	UI	IE11 브라우저를 지원해야 하는 상황에서 추천하는 라이브러리 primefaces.org 참조
@reduxjs/toolkit	앱 내부 저장소 (store), API 연동	redux 는 전체 react 프로젝트의 절반 정도에 쓰이는 저장소 라이브러리인데 react 컴포넌트 외부에서 컴포넌트 직접 접근을 위해 쓴다. redux 외에도 recoil, mobX 등 다른 라이브러리가 많이 나오므로 프로젝트 상황에 맞게 검토하여 사용하면 된다. react + thunk 조합으로 api 연동을 하는 것이 많이 쓰이는 방식이다. API 연동 라이브러리는 redux/toolkit

라이브러리	사용이유	비고
		에 내장된 RTK Query 를 사용하였다. react-query 도 검토했으나 redux 와 같이 사용시 저장소가 2개가 생기고 사용하다 보면 저장소가 애매하게 섞 이는 등 문제가 발생한다는 사례를 듣 고 사용하지 않기로 했다.
chart.js react-chartjs-2	차트, 그래프 UI	chart.js를 사용하기 위한 라이브러리
typescript	typescript 사용	javascript의 superset, javascript에 타입을 추가한 형태 *예 name:string = " *도입 효과 1. 엄격한 type 값을 지켜야 하므로 에러 방지 2. 작성자의 의도를 쉽게 알 수 있어 유지보수하기 좋음
tinymce	문서 편집	오픈소스 Text Rich Editor
cloud-portal-design-token		디자인 토큰 라이브러리

2.5.5. 환경변수 선언

프로젝트 전체에서 공통적으로 참조할 api url 등은 환경변수 파일(.env) 에 기술한다.

원래 CRA(Create React App)로 제작한 환경변수 파일명은 env.prd, env.dev, env.local 등
으로 사용하게 되어있어 특별한 경우가 아니면 그대로 쓰면 된다.

클라우드 포탈의 경우 환경변수 이름을 커스터마이징 해야 하는 요구사항이 있어
package.json 파일에 추가 설정을 한다.

.env.loc (로컬 정보)

```

REACT_APP_TITLE =CLOUD_PORTAL
REACT_APP_API_URL=http://localhost
REACT_APP_SSO_AUTH=http://localhost/sso
REACT_APP_SSO_LOGOUT=http://localhost/logout
REACT_APP_REDIRECT_URI=http://localhost/auth/callback
REACT_APP_FILE_UPLOAD_URL=http://localhost/file/upload
REACT_APP_FILE_DOWNLOAD_URL=http://localhost/file/download
REACT_APP_FILE_ALL_DOWNLOAD_URL=http://localhost/file/downloadAll
REACT_APP_FILE_GETIMG_URL=http://localhost/file/getImg
REACT_APP_USER_URL=http://localhost:3000
REACT_APP_ADMIN_URL=http://localhost:3001
REACT_APP_SURVEY_URL=http://localhost:3002
REACT_APP_JIRA_URL=http://10.40.32.24:8010
REACT_APP_DASHBOARD_DEV_URL=http://10.40.2.10:5601/
REACT_APP_DASHBOARD_PRD_URL=http://10.40.20.10:5601/
REACT_APP_IAM=CLP_USR
REACT_APP_MODE=local

```

.env.dev (개발 서버 정보)

```

REACT_APP_TITLE=CLOUD_PORTAL
REACT_APP_API_URL=http://10.40.0.10:28080/clp
REACT_APP_SSO_AUTH=http://10.40.0.10:28080/clp/sso
REACT_APP_SSO_LOGOUT=http://10.40.0.10:28080/clp/logout
REACT_APP_REDIRECT_URI=http://10.40.0.10:28080/clp/auth/callback
REACT_APP_FILE_UPLOAD_URL=http://10.40.0.10:28080/clp/file/upload
REACT_APP_FILE_DOWNLOAD_URL=http://10.40.0.10:28080/clp/file/download
REACT_APP_FILE_ALL_DOWNLOAD_URL=http://10.40.0.10:28080/clp/file/downloadAll
REACT_APP_USER_URL=http://10.40.0.10:28080
REACT_APP_ADMIN_URL=http://10.40.0.10:28081
REACT_APP_SURVEY_URL=http://10.40.0.10:28082
REACT_APP_JIRA_URL=http://10.40.32.24:8010
REACT_APP_DASHBOARD_DEV_URL=http://10.40.2.10:5601/
REACT_APP_DASHBOARD_PRD_URL=http://10.40.20.10:5601/
REACT_APP_IAM=CLP_USR
REACT_APP_MODE=development

```

.env.tst (스테이징 서버 정보)

```

REACT_APP_TITLE =CLOUD_PORTAL
REACT_APP_API_URL=http://localhost
REACT_APP_SSO_AUTH=http://localhost/sso
REACT_APP_SSO_LOGOUT=http://localhost/logout
REACT_APP_REDIRECT_URI=http://localhost/auth/callback
REACT_APP_FILE_UPLOAD_URL=http://localhost/file/upload
REACT_APP_FILE_DOWNLOAD_URL=http://localhost/file/download
REACT_APP_FILE_ALL_DOWNLOAD_URL=http://localhost/file/downloadAll
REACT_APP_USER_URL=http://localhost:3000
REACT_APP_ADMIN_URL=http://localhost:3001
REACT_APP_SURVEY_URL=http://localhost:3002
REACT_APP_JIRA_URL=http://10.40.32.24:8010
REACT_APP_DASHBOARD_DEV_URL=http://10.40.2.10:5601/
REACT_APP_DASHBOARD_PRD_URL=http://10.40.20.10:5601/
REACT_APP_IAM=CLP_USR
REACT_APP_MODE=test

```

.env.prn (운영 서버 정보)

```

REACT_APP_TITLE=CLOUD_PORTAL
REACT_APP_API_URL=http://10.40.16.6:18080/clp
REACT_APP_SSO_AUTH=http://10.40.16.6:18080/clp/sso
REACT_APP_SSO_LOGOUT=http://10.40.16.6:18080/clp/logout
REACT_APP_REDIRECT_URI=http://10.40.16.6:18080/clp/auth/callback
REACT_APP_FILE_UPLOAD_URL=http://10.40.16.6:18080/clp/file/upload
REACT_APP_FILE_DOWNLOAD_URL=http://10.40.16.6:18080/clp/file/download
REACT_APP_FILE_ALL_DOWNLOAD_URL=http://10.40.16.6:18080/clp/file/downloadAll
REACT_APP_USER_URL=http://10.40.16.6:18080
REACT_APP_ADMIN_URL=http://10.40.16.6:18081
REACT_APP_SURVEY_URL=http://223.130.168.171:443
REACT_APP_JIRA_URL=http://10.40.32.24:8010
REACT_APP_DASHBOARD_DEV_URL=http://10.40.2.10:5601/
REACT_APP_DASHBOARD_PRD_URL=http://10.40.20.10:5601/
REACT_APP_IAM=CLP_USR
REACT_APP_MODE=production

```

위 환경변수를 쓸 수 있도록 먼저 env-cmd를 설치한다.

```
npm install env-cmd
```

package.json 의 "scripts" 섹션에 추가 설정을 한다.

package.json

```
"scripts": {
  "start": "env-cmd -f .env.loc react-scripts start",
  "dev": "env-cmd -f .env.dev react-scripts build",
  "tst": "env-cmd -f .env.tst react-scripts build",
  "prd": "env-cmd -f .env.prd react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
},
```

프로젝트 실행할 때 위 scripts 내의 명령어를 npm run을 사용하여 실행할 수 있다.
npm run start, npm run start:dev, npm run build:tst 등으로 실행 가능하다.

소스코드 내에서 .env 파일 내의 환경변수를 호출하는 방법은 아래와 같다.

```
process.env.REACT_APP_API_URL
```

소스코드에서 환경변수 자동완성을 위한 추가 설정을 하려면 /src/react-app-env.d.ts 파일 안에 아래와 같이 소스코드를 덧붙인다.

react-app-env.d.ts (NODE_ENV, PUBLIC_URL 은 수정불가)

```
/// <reference types="react-scripts" />

declare namespace NodeJS {
  interface ProcessEnv {
    NODE_ENV: 'development' | 'production' | 'test';
    PUBLIC_URL: string;
    REACT_APP_TITLE: string;
    REACT_APP_API_URL: string;
    REACT_APP_MODE: string;
  }
}
```

빌드 명령어에 따른 파일의 우선순위는 아래와 같다.

```
npm start: .env.dev > .env
npm run build: .env.prd > .env
npm test: .env.tst > .env
```

모든 .env 파일은 중첩이 되며 왼쪽 파일이 우선순위가 높다. 즉 똑같은 환경변수가 선언

되어 있는 경우 우측의 .env 파일의 규칙이 가장 우선순위가 낮다.

2.6. 개발 디렉토리 구조

1depth	2depth	3depth	4depth	설명
build				빌드 결과 소스
node_modules				npm 라이브러리
public	index.html			index.html 포함하는 가상 DOM을 위한 파일
src				
	app			
		core		api 연동 관련 소스
			models	api 통신할 데이터의 interface 또는 type
			services	api 연동
			slices	redux의 slice 파일 모음
		pages		페이지 컴포넌트
		routes		라우터
		shared		공유 자원 모음
			components	공유용 컴포넌트들
			config	공통 설정파일 모음
			hooks	공유용 hook 모음
			layouts	페이지 레이아웃
			utils	유틸 모음
	assets			디자인 리소스(images, css)

2.7. Front-End 계층구조

2.7.1. API 계층

API 연동과 관련된 기능 정의된 파일이 있는 곳.

src > app > core > models, services

2.7.2. Assets 계층

멀티미디어 파일과 같이 소스코드가 아닌 파일 (이미지, css)을 모아놓은 곳.

src > assets

2.7.3. Shared 계층

공유용 컴포넌트와 util 함수들을 저장하는 곳.

src > app > shared > components, config, hooks, layouts, utils

2.7.4. Store 계층

Redux, Vuex 등 상태관리 라이브러리를 사용할 경우 필요한 곳. 관련된 파일들을 모아둠.

src > app > core

src > app > core > slices

2.7.5. Route 계층

라우팅 파일을 모아놓은 계층

src > app > routes

2.7.6. View 계층

구성된 모든 페이지를 저장하는 최상위의 계층.

위에 설명된 다른 계층의 파일들을 import 하여 쓸 수 있는 유일한 계층.

src > app > pages

src > App.tsx, index.tsx

3. 명명 표준

3.1. 프로젝트

프로젝트명은 EAMS 시스템에 등록된 코드를 기준으로 한다.

3.1.1. 시스템코드

시스템명	시스템코드
IBK클라우드플랫폼	ICP
상시감시시스템	RTM

3.1.2. 업무코드

업무명	업무코드
클라우드포탈	CLP
상시감시시스템	RTM

3.1.3. 프로젝트명

프로젝트명의 기본 형태는 아래와 같다.

[시스템코드]+'-'+[업무코드]+'-'+[서비스구분]+'-'+[단위업무구분]+'-'+

단위업무구분의 경우 업무 특성을 고려하여 별도 제약을 두지 않으며, 생략 가능하다.

업무시스템	단위업무	프로젝트명
클라우드 포탈	사용자 페이지	icp-clp-front-user
	관리자 페이지	icp-clp-front-admin
상시감시시스템	-	rtm-rtm-front

3.2. 화면 디렉토리 명명 규칙

화면 디렉토리의 기본 경로는 \${웹루트}/src/app/pages/ 이다.

화면 디렉토리는 업무코드 Lv1 + 업무코드 Lv2로 구성된다.

예) \${웹루트}/src/app/pages/clp/main

프로젝트명		업무코드 Lv1		업무코드 Lv	
		접근권한		메뉴명 (1depth)	
클라우드포탈	icp-clp-front-user	사용자	USR	메인	MAN
				클라우드소개	INR
				매뉴얼	MNL
				신청하기	ASC
		관리자	MNG	요청관리	RQD
				운영관리	OPT
클라우드포탈 관리자	icp-clp-front-admin	관리자	MNG	사용자관리	URM
				사이트관리	STM
				설문관리	QSM
				매뉴얼관리	MNM
				신청관리	APM
				요청관리	RQM
				운영관리	OPM

3.3. 파일명 명명 규칙

클라우드 프로젝트의 개발 환경은 Windows, 배포 환경은 리눅스이다. **개발 소스의 OS 환경이 바뀌어도 문제없는 명명 규칙인 '소문자-소문자' 형태를 기본으로 한다.**

3.3.1. 폴더명

종류	파일명	예시
폴더명	소문자, 하이픈으로 연결	notice-list

3.3.2. app > core 내 파일들

종류	파일명	suffix	예시
model	소문자, 하이픈 연결	-	sample-util.ts
hook	소문자, 하이픈 연결	.hook	sample-util.hook.ts
slice	소문자, 하이픈 연결	.slice	sample-user.slice.ts
service	소문자, 하이픈 연결	.sertive	sample-list.sertive.ts

3.3.3. app > pages, routes, shared 내 파일들

종류	파일명	suffix	예시
Component	첫 글자는 대문자	-	Sample.tsx

3.4. 화면 컨트롤 명명 규칙

3.4.1. 기본 함수명

종류	함수명
hook	use 로 시작하며 카멜 표기법 사용 use로 시작하면 React에서 이를 hook 으로 인지 ex) useState, useEffect 등
slice	소문자, 카멜 표기법
service	카멜 표기법
Component	대문자로 시작

3.4.2. 변수 선언

변수 선언		
변수 선언	const	소문자
	let	소문자
	var	사용하지 않는다
배열 선언	[{ key : value }, { key : value }, { key : value }]	소문자

3.5. 주석 작성 규칙

- * 주석은 화면 개발 시 작성되며, 화면의 개발 목적, 수정 내용, 세부적인 기능 설명의 기록을 남긴다.
- * 주석 영역은 화면 함수 영역, 구현 주석, 공통 스크립트 주석으로 총 3가지로 구분한다.

3.5.1. React 주석처리

javascript와 동일하게 처리한다.

```
// singleline comments
/*
  multiline
  comments
*/
return (
  { /* comments */ }
)
```

3.5.2. 화면 함수 영역

화면의 기능을 구현하는 Function 단위 별로 작성한다.
별도의 변경 이력 관리는 하지 않는다.

```
/**
 * @함수명 : $$clear
 * @설명 : 컴포넌트의 값을 초기화합니다.
 * @입력 : arg0 - 초기화 할 대상 컴포넌트의 ID
 * @입력 : arg1 - 초기화 입력 값 (옵션)
 * @출력 :
 */
```

3.5.3. 구현 주석 영역

구현 주석은 개발코드에서 코드를 간략히 설명하기 위하여 사용하는 주석을 의미한다.
주석은 장황한 설명보다는 정확하고 간결하게 의미가 전달될 수 있도록 기술한다.
하지만 너무 간략하여 의미전달이 되지 않도록 작성해서는 안된다.

```
ModelUI.clear = function(containerId, withInitValue) {
  // 초기화 입력 값이 null 일 경우 초기화 값을 true로 처리한다.
  if(withInitValue == null) {
    withInitValue = true;
  }
}
```

3.5.4. 공통 스크립트 주석 영역

공통 기능을 구현하는 Function 단위 별로 작성한다.
별도의 변경 이력 관리는 하지 않는다.

```

/**
 * @class
 * @description 메시지(Confirm, Alert, Error) 관련 함수 작성 객체
 */
var imessage = new Object();

/**
 * @description 간단한 메시지 창을 보여준다. (모달)
 * @param message 메시지 창에 표현될 텍스트
 * @param func 메시지 창 종료 시 콜백 함수 정의
 * @param params {title: 메시지 창 제목, okText: 확인 버튼 텍스트, cancelText:
취소 버튼 텍스트}
 * @return
 * @author 홍길동(ns000)
 */
imessage.show = function(message, func, params) {
  // contents..
}

```

4. 퍼블리싱

4.1. 화면 컨트롤 명명 규칙

4.1.1. id, name, label 규칙

구분	컨트롤명	prefix
입력박스	input	ipt
콤보	select	slt
라벨	label	lbl
이미지	image	img
텍스트영역	textarea	txt
버튼	button	btn
서브밋버튼	submit	sbtn
테이블	table	tbl
웹 폼	form	frm

하이픈으로 연결한다. prefix '-' 컨트롤명 '-' (일련번호)

ex) frm-login, btn-select-1

컨트롤명이 중복되는 경우에는 일련번호를 붙인다.

4.1.2. React 기본 제공 컨트롤

구분	prefix	예시
React 기본 제공 hook	React.use	React.useEffect React.useState React.useCallback React.useRef
React 기본 제공 타입	React.	React.FC
Custom hook (개별 제작)	use	useForm

4.1.3. RTK Query

구분	prefix	suffix	예시
조회 (GET, POST 등 가능)	use	Query	useGetIdQuery
등록, 수정, 삭제 (POST, PUT, DELETE)	use	Mutation	usePostMutation

4.1.4. 기타

구분	prefix	예시
이벤트 핸들링하는 함수명	handle	handleChange handleClick handleMouseOver

4.2. 화면 구조

SamplePage.tsx

```
import * as React from 'react';

const [페이지이름]:React.FC = () => { //FC 는 FunctionComponent 의 약자
  //hook 선언 useState, useEffect
  //hook 은 다른 함수 내부나 조건문 안에 선언 불가

  //스크립트 작성
  return(
    <>
      //렌더링 될 UI 템플릿
    </>
  )
}

export default [페이지이름];
```

4.3. 스타일

4.3.1. 디자인 토큰 안내

App.tsx 에 임포트 되어있음
css var에 의존적으로 사용해야 함

Design Token CSS Library
<pre>:root { --primary-color: blue --secondary-color: red }</pre>
Framework > .css
<pre>/* 토큰의 변수 값을 var 형태로 임포트하여 사용한다. */ button { color: var(--primary-color); }</pre>

4.3.2. CSS 작성 규칙

카멜 표기법으로 작성

4.3.3. 이미지 사용 방법

```
//이미지를 import 하여 사용한다
import * as React from 'react';
import logolmg from '../assets/images/logo.png'

const SamplePage:React.FC = () => {
  return(
    <>
      <img src={logolmg} alt="Cloud Portal Admin" />
    </>)
  }
export default SamplePage;
```

4.3.4. class 추가

```
import * as React from 'react';

const SamplePage:React.FC = () => {
  let isDarkmode = false;
  return(
```

```

<BasePage>
  { /* 인라인으로 클래스명 넣기 */ }
  <h1 className="title">Hello, world! </h1>
  { /* 변수로 클래스명 넣기 */ }
  <h1 className={ isDarkmode ? 'light' : 'dark' }>Hello, world! </h1>
</BasePage>)
}
export default SamplePage;

```

4.3.5. 스타일 추가

//기본 스타일 요소를 카멜 표기법으로 호출할 수 있다. 값 영역에 'px' 는 생략

```

import * as React from 'react';

const SamplePage:React.FC = () => {
  let sampleStyle = {
    marginLeft: 20,
    color: 'red'
  }

  return(
    <BasePage>
      { /* 인라인으로 스타일 넣기 */ }
      <div style={{ marginLeft: 20, height: 200 }}></div>
      { /* 변수로 스타일 넣기 */ }
      <h1 style={sampleStyle}>Hello, world! </h1>
    </BasePage>)
  }
export default SamplePage;

```

4.4. UI Library 소개

primereact 오픈 소스 UI 라이브러리(v10.9.2)를 활용한다.
 (2022년 SSO 인증 때문에, Edge 의 ie11 모드 동작 고려하여 선택)
 primefaces.org 에서 자세한 명세 참조할 것.

4.5. 화면 생성 안내

4.5.1. 페이지 생성

src/app/pages/[페이지폴더명]/[페이지이름].tsx 파일 생성

SamplePage.tsx

```
//사용하는 공통 라이브러리 등 import
import * as React from 'react';
import useBasePage from '../shared/hooks/base-page.hook';

const SampleEdit:React.FC = () => {
  //페이지 공통 hook 중 사용할 함수, 변수를 임포트.
  //이렇게 쓰면 goPage('/notice/list') 등 함수를 바로 사용 가능
  const { paramId, goPage, goBack, commonCode, BasePage } = useBasePage()

  const [detail, setDetail] = React.useState(null)

  //리액트의 hook 은 컴포넌트 상단에 선언 (useState, useEffect 등)
  //hook 은 일반 함수 내에서 사용 불가

  //페이지 로딩되고 초기에 한 번만 실행되는 hook
  React.useEffect(()=> {
    //실행할 함수 작성
  }, []) //<-여기를 [] 빈 배열로 놓아야 초기 한 번만 실행됨.
  //아무 값도 없으면 무한 호출되므로 주의

  React.useEffect(()=> {
    //실행할 함수 작성
  }, [ detail ]) //<- 이 배열 안에 변수가 들어있으면, 이 값에 변동이 있을 때마다
  //해당 useEffect hook이 실행된다

  //return 부분에 렌더링될 UI 작성
  //UI 요소는 단일한 컴포넌트가 되도록 <></> 또는 <div></div> 안에 묶어줘야 함
  //클라우드 포탈의 경우 페이지들은 공통 컴포넌트인 <BasePage> 안에 작성
  return(
    <BasePage>
      <h1>Hello, world! </h1>
    </BasePage>)
}

export default SampleEdit;
```

4.5.2. 라우팅 생성

src/app/routes/routes.js 에 생성한 페이지를 추가한다.

sample-routes.js

```
//구조
{
  path: '/sample', //주소창에서 접근할 주소. http://[도메인명]/sample
                //컴포넌트에서는 navigate('/sample') 로 이동
  element: <FullLayout />, //path 로 이동할 컴포넌트 본체

  children: [ //children 내부에 선언한 path 는 / 로 접근
                //예를 들어 아래의 경우 'sample/list' 로 접근 가능
                { path: 'list', name: 'sample list', exact: true, element: <SampleList /> },
              ],
  //위와 같이 정의한 경우 주소창에 이렇게 입력하여 접근 가능
  //http://[도메인명]/sample/list => <SampleList /> 화면
},

import { lazy } from 'react';
import { Navigate } from 'react-router-dom';
import Loadable from '../shared/layouts/loader/Loadable';

/***** Pages *****/
//페이지 컴포넌트명, 경로 추가
const SampleList = Loadable(lazy(async () => await import('../pages/sample/SampleList')));

/*****Routes*****/
const PortalRoutes = (isLoggedIn) => [
  {
    path: '/sample',
    element: <FullLayout />,
    children: [
      { path: 'list', name: 'sample list', exact: true, element: <SampleList /> },
      { path: 'detail/:id', name: 'sample detail', exact: true, element: <SampleDetail /> },
      { path: 'edit/:id', name: 'sample SampleEdit', exact: true, element: <SampleEdit /> },
    ],
  },
  {
    path: '*', //라우터에 정의되지 않은 나머지 path 의 경우에 404 페이지로 리다이렉트 한다
    element: <Navigate to="/auth/404" />
  },
];
```


sample-routes.js

```
];

export default PortalRoutes;
```

4.6. 화면 이동

React 함수 컴포넌트에서는 useNavigator() hook을 사용하여 페이지 이동을 처리한다.

- 페이지 이동 : navigator(라우터 경로(예: "/notice/list"))
- 뒤로가기 :navigator(-1)

4.6.1. 특정 페이지로 이동

SamplePage.tsx

```
const SamplePage:React.FC = () => {
  const { goPage, goBack } = useBasePage()

  const goListPage = () => {
    goPage('/sample/list')
  }

  const cancel = () => {
    //뒤로가기
    goBack();
  }

  return(
    <BasePage>
      <div className='detailFooterButton'>
        <Button onClick={cancel}>취소</Button>
        <Button onClick={goListPage}>목록으로 이동</Button>
      </div>
    </BasePage>)
}

export default SampleEdit;
```

//화면 템플릿에서는 link 사용

```
const url = '/sample/list'
...
<Link to={ url }>이동</Link>
```

4.6.2. 특정 페이지에 데이터와 함께 이동

데이터를 보내는 페이지
<pre>const SamplePage:React.FC = () => { const { goPageWithData } = useBasePage() const cancel = () => { goPageWithData('/man', { id: 's12345' }); } } export default SampleEdit;</pre>
데이터를 받는 페이지
<pre>const SamplePage:React.FC = () => { const { location } = useBasePage() const state = location?.state as { id: string }; //as ~ 부분은 데이터 넣어준 곳과 같은 타입으로 정의 console.log('GET DATA =>', state?.id)</pre>

4.7. 공유용 컴포넌트 생성 안내

생성 기준	2군데 이상에서 호출되는 반복적인 UI는 독립된 컴포넌트로 생성하여 재사용하는 것을 추천한다.
생성 위치	src/app/shared/components src/app/shared/index.ts 에 컴포넌트 파일명 추가 (모듈이 많아질 경우 관리 용이)
유의사항	2 depth 이상의 복잡한 컴포넌트 구조는 권장하지 않는다. (지나치게 깊은 구조는 핸들링하기 어려움)

4.8. 공유용 custom hook 생성 안내

생성 기준	2군데 이상에서 반복적으로 호출되는 기능을 필요시에
-------	------------------------------

	hook 으로 생성한다.
생성 방법	함수명, 파일명 앞에 use 키워드를 붙여 생성하면 React 가 이를 hook 으로 인식한다.
이름 규칙	카멜 표기법 ex) useLoginCase
생성 위치	src/app/shared/hooks src/app/shared/index.js 에 hook 파일명 추가 (모듈이 많아질 경우 관리 용이)
사용방법	함수 컴포넌트 내부에서 호출 let getName = useGetName()
제약사항	<ul style="list-style-type: none"> - hook 은 함수 컴포넌트에서만 호출 가능하다. - .tsx 파일의 최상단에서 호출해야 한다. - hook 은 조건문 안에서 호출할 수 없다. - React 일반 hook 안에서 custom hook을 호출할 수 없다. 예를 들어 useEffect 안에서 useBasePage를 호출할 수 없다. - custom hook 안에서는 React 의 hook 호출 가능하다.

4.9. 유의사항

4.9.1. 함수

리액트 렌더링 엔진은 UI Dom Tree를 업데이트 할 때 바뀐 변수가 있는지 빠르게 훑는 얇은 비교를 수행한다. (변수 안의 내용물을 일일이 검사하지 않는다.) UI를 업데이트 할 때, 새로운 변수를 리턴해 주어서 해당 부분이 바뀌었음을 리액트 렌더링 엔진에 알려줘야 한다. 반복문 내에서 UI를 만들 때는 새로운 배열을 리턴해주는 **map, filter, slice, reduce** 를 사용한다. push, pop 등은 UI 업데이트와 직접적인 연관이 없는 비즈니스 로직 작성에만 사용한다. 배열에 내용을 추가 할 때는 push 대신 concat을, object 에 내용을 추가할 때는 assign 표현식을 사용한다.

4.9.2. 성능

모든 반복문에 key 사용해 주어야 한다. (위치 변동시 리렌더링을 막는다)
그밖에 리액트 개발 문서의 useMemo, useCallback를 참조하여 필요한 곳에 사용한다.

4.9.3. 크로싱 브라우저

Edge 브라우저의 IE11 모드에서 화면이 돌아가는지 확인

4.9.4. HTML 표준 준수

- * html 이 깨지면 react에서 에러가 나므로 개발자 도구를 켜놓고 에러가 나는지 보면서 퍼블리싱을 해야 한다.
- * SPA 는 단일 html 이라는 특성상 하나의 컴포넌트에서 에러가 날 때 다른 컴포넌트의

(예) <table> 아래 바로 <tr>을 넣으면 안됨 (tbody 아래에 tr 넣어야 함)

```
<table>
  <tbody><tr><td>...
```

동작에 영향을 미치므로 사소한 에러가 생기는 것도 주의해야 한다.

4.9.5. 동영상 사용

typescript 라서 타입 에러 등으로 동영상 파일 추가가 안 될 경우, 동영상이 들어있는 폴더 안에 videos.d.ts 파일을 추가하고 필요한 동영상 확장자를 타입으로 추가해준다.

src/assets/videos/videos.d.ts
declare module '*.mp4';
tsconfig.json
<pre>{ "compilerOptions": { ... }, "typeRoots": ["videos"] //비디오가 들어있는 폴더명 추가 }</pre>
동영상 불러오는 화면에서 영상 사용
<pre>import video1 from '.././.././../assets/videos/main1-cityview.mp4'; ... <video muted autoPlay loop className='videoBg'> <source src={video1} type='video/mp4' ></source> </video></pre>

5. 개발

5.1. 화면ID 명명 규칙

문서관리표준 문서의 관리번호 부여 규칙을 따른다.

[업무코드]+'-'+[메뉴명약어3자리]+'-'+[화면구분]+'-'+[일련번호3자리]+'-'+

화면목록 참고

업무코드 Lv1	업무코드 Lv2 (메뉴 대분류)	업무코드 Lv3 (메뉴명약어3자리)	비고
----------	----------------------	------------------------	----

업무코드 Lv1	업무코드 Lv2 (메뉴 대분류)	업무코드 Lv3 (메뉴명약어3자리)	비고
예)관리자	/사이트관리	/공지사항	
mng	/stm	/ntc/CLPNTCM91010.tsx	목록
		/ntc/CLPNTCM91020.tsx	상세보기/등록/수정
		/ntc/CLPNTCM910.css	

5.2. 구성

5.2.1. 디자인 에디터

VSCode 사용

5.2.2. 구성

SamplePage.tsx

```
import * as React from 'react';

const [페이지이름]:React.FC = () => { //FC 는 FunctionComponent 의 약자
  //hook 선언 useState, useEffect
  //hook 은 다른 함수 내부나 조건문 안에 선언 불가

  //javascript

  return(
    <>
      //렌더링 될 UI 템플릿
    </>
  )
}
```

export default [페이지이름];

5.2.3. 실행 순서 ([별첨1] 참조)

index.tsx -> App.tsx -> routes 화면 분기 -> auth check (로그인/권한 체크) -> FullLayout의 <Outlet />으로 이동

5.3. 시작

5.3.1. 화면 컨트롤 초기화

함수 컴포넌트에는 클래스형 컴포넌트와 다르게 별도의 lifecycle 이 없다.
대신 useEffect hook을 초기화에 쓸 수 있다.

```
import * as React from 'react';

const SampleEdit:React.FC = () => {
  //페이지 로딩되고 초기에 한 번만 실행되는 hook
  React.useEffect(()=> {
    //실행할 함수 작성
  }, []) //<-여기를 [] 빈 배열로 놓아야 초기 한 번만 실행됨.
    //아무 값도 없으면 무한 호출되므로 주의

  React.useEffect(()=> {
    //실행할 함수 작성
  }, [ detail ]) //<- 이 배열 안에 변수가 들어있으면, 이 값에 변동이 있을 때마다
    //해당 useEffect hook이 실행된다

  return(
    <BasePage>
      <h1>Hello, world! </h1>
    </BasePage>)
}

export default SampleEdit;
```

5.4. 데이터 처리

5.4.1. Formatting

primereact의 InputMask 컴포넌트를 이용한다.

마스킹

```
//Basic
<InputMask id="basic" mask="99-999999" value={val1} placeholder="99-999999"
onChange={(e) => setVal1(e.value)} />

//SSN
<InputMask id="ssn" mask="999-99-9999" value={val2} placeholder="999-99-9999"
onChange={(e) => setVal2(e.value)} />

//Date
<InputMask id="date" mask="9999/99/99" value={val3} placeholder="9999/99/99"
slotChar="yyyy/mm/dd" onChange={(e) => setVal3(e.value)} />
```

```

//Phone
//mask 안에 정의한 형식으로 바뀐다
<InputMask id="phone" mask="(999) 9999-9999" value={val4}
  placeholder="(999) 9999-9999" onChange={(e) => setVal4(e.value)} />
<InputMask id="phone" mask="999-9999-9999"
  value={val4} placeholder="999-9999-9999" onChange={(e) => setVal4(e.value)} />

//Serial
<InputMask id="serial" mask="a*-999-a999" value={val6}
  placeholder="a*-999-a999" onChange={(e) => setVal6(e.value)}> </InputMask>

//숫자 인풋 마스크
//ex)123,456,789
<InputNumber inputId="integeronly" value={value1}
  onValueChange={(e) => setValue1(e.value)} />

//prefix, suffix 붙는 경우
// 132 mi
<InputNumber inputId="mile" value={value13}
  onValueChange={(e) => setValue13(e.value)} suffix=" mi" />

// %123
<InputNumber inputId="percent" value={value14}
  onValueChange={(e) => setValue14(e.value)} prefix="%" />

```

5.4.2. Validation (유효성 체크)

save 등 값을 보내거나 수정하는 함수 내에서 setState 로 받아온 값들을 체크하여 보내는 것이 일반적이고, 좀 더 간편한 작업을 위해 Formik, Yup 사용을 추천한다.

먼저 formik, yup 을 설치한다.

```
npm install formik yup
```

유효성 체크 페이지 예시

```

import * as Yup from 'yup';
import { useFormik } from 'formik';

//유효성체크 위한 formik
const formik = useFormik({

```

```

initialValues: values,
enableReinitialize: true,
validationSchema: Yup.object({
  //사용자ID
  wrtnEmn: Yup.string()
    .required(commonMsg.COM_ERR_MSG_001)
    .max(6, '6'+commonMsg.COM_ERR_MSG_003),
  //성명
  emm: Yup.string()
    .required(commonMsg.COM_ERR_MSG_001)
    .max(50, '50'+commonMsg.COM_ERR_MSG_003),
  //이메일주소
  ead: Yup.string().trim()
    .nullable()
    .required(commonMsg.COM_ERR_MSG_001)
    .email('유효하지 않은 이메일 형식입니다.')
    .max(50, '50'+commonMsg.COM_ERR_MSG_003),
}),
onSubmit: values => {
  updatePost(values).unwrap()
  .then(( response:any ) => {
    console.log('updatePost response =>',response)
    if(response) {
      confirmDialog({
        message: '접속권한 신청정보가 수정되었습니다.',
        className: 'noHeader oneButton',
        acceptLabel: '확인',
        accept: ()=> {
          setMode('view')
          goPage(`/apm/caa/${paramId}`);
        },
      })
    }
  }).catch((error) => console.error('rejected', error))
}
})
...

const isFormikError = (name: any) => {

```



```

    return Boolean(formik.touched[name] && formik.errors[name])
  }
  const contentsInfo = {
    title: '신청 내용',
    mode: mode,
    hasRequired: true,
    rows: [
      {
        cols: [
          {
            required: true,
            key: '신청구분',
            error: isFormikError('cctnAthrAplcTcd'),
            helperText: formik.errors.cctnAthrAplcTcd,
            value: <span> {detail?.cctnAthrAplcTcdNm}</span>,
            editingValue: (
              CLD_POTL_APLC_TCD?.map((category) => {
                return (
                  <span
                    key={category.value}
                    className="field-radiobutton mr20">
                      <Checkbox
                        inputId={category.value}
                        name='cctnAthrAplcTcd'
                        value={category.value}
                        className='mt7'
                        onChange={(e) => {
                          formik.handleChange(e)
                        }}
                      />
                    <label className='ml5'
                      htmlFor={category.value}>
                        {category.name}
                      </label>
                    </span>
                  )
                )
              })
            ),
          },
        ],
      },
    ],
  },

```

```

    ]
    },
    ...

return(
  <BasePage>
    { /* 신청 내용 */ }
    <form onSubmit={formik.handleSubmit}>
      <ViewTemplate {...contentsInfo} />

```

5.4.3. Date

moment.js 사용. 자세한 사용법은 moment.js 공식 문서 참조

```

if(moment.isDate(event.target.value)) {
  value = moment(event.target.value).format('YYYYMMDD')
}

```

5.5. 컨트롤

자세한 예시는 DSP 화면가이드 메뉴를 통해 확인할 수 있다.

5.5.1. Input

```

const [value1, setValue1] = React.useState("");
return (<InputText value={value1} onChange={(e) => setValue1(e.target.value)} />)

```

Basic

```

<InputTextarea value={value1} onChange={(e) => setValue1(e.target.value)}
  rows={5} cols={30} />

```

Basic

5.5.2. Select

```

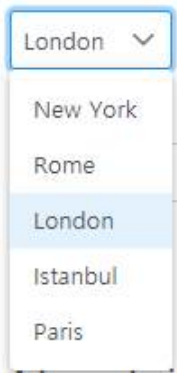
const [selectedCity1, setSelectedCity1] = React.useState<any>(null);

```

```
const cities = [
  { name: 'New York', code: 'NY' },
  { name: 'Rome', code: 'RM' },
  { name: 'London', code: 'LDN' },
  { name: 'Istanbul', code: 'IST' },
  { name: 'Paris', code: 'PRS' }
];

return(
  <Dropdown value={selectedCity1} options={cities}
    onChange={onCityChange} optionLabel="name" placeholder="Select a City" />)

```



5.5.3. Multiple Select

```
const [selectedCities1, setSelectedCities1] = React.useState(null);

const cities = [
  { name: 'New York', code: 'NY' },
  { name: 'Rome', code: 'RM' },
  { name: 'London', code: 'LDN' },
  { name: 'Istanbul', code: 'IST' },
  { name: 'Paris', code: 'PRS' }
];

return(
  <MultiSelect value={selectedCities1} options={cities}
    onChange={(e) => setSelectedCities1(e.value)}
    optionLabel="name" placeholder="Select a City" maxSelectedLabels={3} />)

```



5.5.4. List

```
const lists = [
  {key:'12313', name:'1'},
  {key:'65488', name:'2'},
  {key:'35765', name:'3'},
  {key:'15368', name:'4'},
]

return (
  <ul>{lists.map( (item, index) =>
    //고유한 key 값을 넣어줘야 성능상 좋다. 가령 순서가 바뀔 때
    //리렌더링 등이 일어나지 않는다. key 값으로 index를 쓰는 건 지양한다.
    <li key={item.key}>{item.name}</li>
  )}
</ul>
)
```

Basic

```
1
2
3
4
```

5.5.5. Datatable

src/app/pages/SampleList.tsx (전체 코드는 이곳에서 참조)

```
import * as React from 'react';
```

```
import { Button, Column, DataTable } from 'primereact';
import {
  useGetPostsQuery,
} from '../core/services/post.service'
import useBasePage from '../shared/hooks/base-page.hook';
import { paginator } from '../shared/utils/base-table';
import { TableHead } from '../core/models/table-head';
```

//테이블 헤더 내용 정의 (퍼블리셔가 1차적으로 작성)

```
const headCells: TableHead[] = [
  {
    field: 'id', //참조하는 값
    header: 'ID', //테이블 헤더에 표출되는 제목
    sortable: false, //sorting 관련내용 추후 업데이트
    style: { width: '7%' } //테이블 너비
  },
  {
    field: 'subject',
    header: '제목',
    sortable: false,
    style: { width: '25%' }
  },
  {
    field: 'file',
    header: '첨부파일',
    sortable: false,
    style: { width: '10%' }
  },
  {
    field: 'hit',
    header: '조회수',
    sortable: false,
    style: { width: '10%' }
  },
  {
    field: 'date',
    header: '등록일',
    sortable: false,
    style: { width: '15%' }
  },
],
```

```

];

const SampleList:React.FC = () => {
  const { goPage, BasePage } = useBasePage()
  const [boardRow, setBoardRows] = React.useState([])
  const [first, setFist] = React.useState(0); //읽어올 데이터 index 시작점
  const [rows, setRows] = React.useState(10); //표시할 행 개수

  const handleCustomPage = (event) => {
    setPageNo(event.first);
    setPageSize(event.rows);
  }

  //게시물 조회조건
  let param = {
    pageSize:10,
    pageNo:1,
    pageGrpInqYn : 'Y',
    pageGrpNbi : 5,
    cldPotlBlbrDcd:'10',
    cldPotlBlbrTcd:''
  }

  //목록조회 API 사용하여 데이터 조회
  const { data:posts, isFetching, isLoading } = useGetPostsQuery( param )

  React.useEffect(()=> {

    if(posts) {
      let response:any = posts
      let boardData = response.contents

      //rows 데이터 배열 초기화
      setBoardRows([])

      //데이터 읽어와서 필요한 데이터만 골라서 테이블 배열에 추가
      boardData.map(({blbrOtptNo, blbrTtlNm, blbrInqNbi, blbrCretTs, wrtnEmn}) => {
        let obj = {
          id: blbrOtptNo,           //번호
          subject: blbrTtlNm,      //제목

```

```

        file: "", //첨부파일
        hit: blbrInqNbi, //조회수
        date: blbrCretTs.join('-') //날짜
    };

    //rows 데이터 업데이트
    setBoardRows((prev) => (
        [...prev, obj]
    ))
    })
}

}, [posts]) //posts 값에 뭔가가 들어오거나 변경될 때 이 useEffect 실행됨

const handleRowClick = (e:any) => {
    let id = e.data.id;
    goPage(`/sample/detail/${id}`);
}

return (
    <BasePage>
        <div className='heading'>
            <h1>게시판 관리 (샘플)</h1>
        </div>
        <div>
            <Button onClick={()=>{goPage(`/sample/register`)}}>등록</Button>
        </div>
        <DataTable
            value={boardRow}
            paginator
            paginatorTemplate={paginator}
            first={first}
            rows={rows}
            onPage={handleCustomPage}
            responsiveLayout='scroll'
            onRowClick={handleRowClick}
        >
            {headCells.map(({field, header, sortable, style}, index) => (
                <Column key={field} field={field} header={header}
                    sortable={sortable} style={style}></Column>
            ))}
        </DataTable>
    </BasePage>
)

```

```

    )}
  </DataTable>
</BasePage>
);
}
export default SampleList;

```

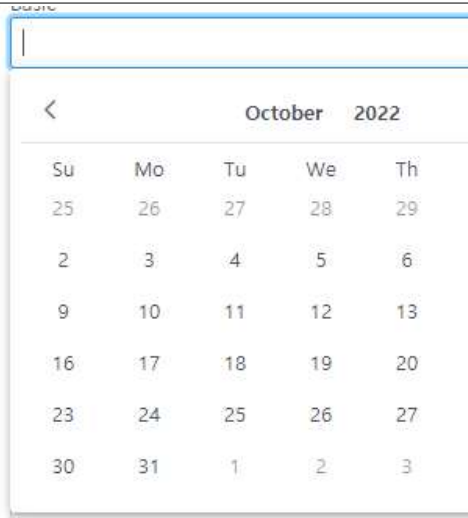
5.5.6. DatePicker

```

const [date1, setDate1] = React.useState<Date | Date[] | undefined>(undefined);

return (
  <Calendar id="basic" value={date1} onChange={(e) => setDate1(e.value)} />
)

```



5.5.7. TextEditor

등록
<pre> import TextEditor from "../../shared/components/ui/text-editor/TextEditor"; //공통컴포넌트 사용 <TextEditor name='qstrCon' className={iff(isFormikError('qstrCon') , 'p-invalid' , '')} value={formik.values?.qstrCon} onChangeText={(e) => formik.setFieldValue('qstrCon', CommonUtils.nullToString(e.htmlValue))} /> </pre>
조회
<pre> <div className='ql-editor qlViewMode' </pre>


```
dangerouslySetInnerHTML={{ __html: detail?.svcSprnCon }}/>
```

5.5.8. Event

기본적으로 javascript에서 쓰는 것과 동일한 이벤트들이고, JSX(TSX) 문법에서는 이를 카멜 케이스로 표기하면 된다.

javascript, JSX, TSX 동일	
<pre>//함수 const handleChange = (event) => { //event.target... } const handleClick = (event) => { } const getID = (event, id) => { }</pre>	
javascript	JSX, TSX
<pre>//템플릿 구현부 //일반 스크립트 작성시 (비교를 위해) <button onChange="handleChange()"> <button onClick="handleClick()"></pre>	<pre>//템플릿 구현부 //함수명 뒤에 () 를 붙이지 않는다 <button onChange={handleChange}> <button onClick={handleClick}></pre>
	<pre>//인라인으로 호출시 <button onChange={ (event) => { getID(event, id)}} onClick = { handleClick } ...</pre>

5.5.9. Chart

<pre>//1. 차트 라이브러리 설치 npm install chart.js react-chartjs-2</pre>
<pre>//2. 화면에서 차트 라이브러리 사용 import { Chart } from "primereact"; import * as React from "react"; import { BasePage } from "../../shared/components/base/BasePage"; const ChartGuide: React.FC = () => {</pre>

```

const [chartData] = React.useState({
  labels: ['A', 'B', 'C'],
  datasets: [ { data: [300, 50, 100],
    backgroundColor: ["#42A5F5", "#66BB6A", "#FFA726"],
    hoverBackgroundColor: ["#64B5F6", "#81C784", "#FFB74D"] } ]
});
const [lightOptions] = React.useState({
  plugins: { legend: { labels: { color: '#495057' } } }
});
const [basicData] = React.useState({
  labels: ['January', 'February', 'March', 'April', 'May', 'June', 'July'],
  datasets: [ { label: 'My First dataset', backgroundColor: '#42A5F5',
    data: [65, 59, 80, 81, 56, 55, 40] },
    { label: 'My Second dataset', backgroundColor: '#FFA726',
    data: [28, 48, 40, 19, 86, 27, 90] } ] });

const [lineStylesData] = React.useState({
  labels: ['January', 'February', 'March', 'April', 'May', 'June', 'July'],
  datasets: [
    { label: 'First Dataset', data: [65, 59, 80, 81, 56, 55, 40],
    fill: false, tension: .4, borderColor: '#42A5F5' },
    { label: 'Second Dataset', data: [28, 48, 40, 19, 86, 27, 90],
    fill: false, borderDash: [5, 5], tension: .4, borderColor: '#66BB6A' },
    { label: 'Third Dataset', data: [12, 51, 62, 33, 21, 62, 45],
    fill: true, borderColor: '#FFA726',
    tension: .4, backgroundColor: 'rgba(255,167,38,0.2)' } ]});
let horizontalOptions = {
  indexAxis: 'y', maintainAspectRatio: false, aspectRatio: .8,
  plugins: { legend: { labels: { color: '#495057' } } },
  scales: { x: { ticks: { color: '#495057' }, grid: { color: '#ebedef' } },
    y: { ticks: { color: '#495057' }, grid: { color: '#ebedef' } } } };
return { basicOptions, horizontalOptions, }
}
const { basicOptions, horizontalOptions } = getLightTheme();

return(
  <BasePage>
    <div>
      <Chart type="pie" data={chartData} options={lightOptions} />
      <Chart type="doughnut" data={chartData} options={lightOptions} />
    </div>
  </BasePage>
)

```

```


    <h5>Vertical</h5>
    <Chart type="bar" data={basicData} options={basicOptions} />
    <h5>Horizontal</h5>
    <Chart type="bar" data={basicData} options={horizontalOptions} />
    <h5>Basic</h5>
    <Chart type="line" data={basicData} options={basicOptions} />
    <h5>Line Styles</h5>
    <Chart type="line" data={lineStylesData} options={basicOptions} />
  </div>
</BasePage>)
}
export default ChartGuide

```

5.5.10. Hook

React에서 함수형 컴포넌트가 클래스형 컴포넌트의 기능을 사용할수 있도록 해주는 기능.
use- 로 시작

useState



```

const [value, setValue] = React.useState("");

<InputText
  value={value}
  onChange={(e) => setValue(e.target.value)} />

```

useEffect

```

//초기화
React.useEffect(() => {
  //실행할 내용 작성
}, []) //<-여기를 [] 빈 배열로 놓아야 초기 한 번만 실행됨.
      //아무 값도 없으면 무한 호출되므로 주의

//변수 변경시 호출
let detail = ""
let user = ""
React.useEffect(() => {

```

```

//실행할 내용 작성
}, [ detail, user ]) //<- 이 배열 안에 변수가 들어있으면, 이 값에 변동이 있을 때마다
//해당 useEffect hook이 실행된다

//clean up
//추후 예시 업데이트

```

Custom Hook - 구현부 (src/app/shared/hooks/base-page.hook.ts 예시 참조)

```

import { useParams } from 'react-router-dom';
import { useNavigate } from 'react-router-dom';
import * as commonCode from '../shared/config/commonCode';
import { BasePage } from '../shared/components/base/BasePage';

//필요한 입력값 정의
const useBasePage = ( sortBy?:string ) => {

  const paramId = params.id;

  //url 라우터로 이동
  const goPage = ( url:string ) => {
    return navigate(url);
  }

  //뒤로 가기
  const goBack = () => {
    return navigate(-1);
  }

  //이 커스텀 훅을 사용하는 곳에서 사용할 변수, 함수, 템플릿을 return 한다.
  return {
    //basePage
    queryParam, paramId,
    commonCode,
    goPage,
    goBack,

    //공통 페이지 UI Component
    BasePage,
  }
}

```

Custom Hook - 구현부 (src/app/shared/hooks/base-page.hook.ts 예시 참조)

```
}
export default useBasePage;
```

Custom Hook - 사용부

//2가지 방법으로 사용가능

//1. 필요한 변수 등을 비구조화 할당으로 직접 가져와서 사용

```
const SampleDetail:React.FC = () => {
  const { paramId, goPage, BasePage } = useBasePage()
  ...
  goPage();
```

//2. 간접 참조 방식으로 사용

```
const SampleDetail:React.FC = () => {
  const basePage = useBasePage()
  ...
  basePage.goPage();
  basePage.paramId = ...
```

.css

```
.show {
  display: block;
}
```

```
.hide {
  display: none;
}
```

.tsx

//row.showIf 가 true 이면 보여주고 아니면 숨긴다

```
<div className={ row?.showIf === false ? 'hide':'show' }></div>
```

5.6. 컨트롤 조작

5.6.1. 화면 컨트롤 접근

```
const floatingEl = React.useRef(null);
...
<div className='floating' ref={floatingEl} ></div>
```

5.6.2. Event

```
const selectCategory = (prop:any, value:string) => {
  ...
}
const handleChange = (e) => { }
<Button onClick={(e) => selectCategory('cldPotlCtgyDcd', category.value)}>
<Checkbox onChange={(e) => { handleChange(e) } } />
```

5.6.3. 엔터키 이벤트

```
const searchKeyDown = (e) => {
  if(e.key === 'Enter') {
    //enter 이벤트 발생시 처리할 동작
  }
}
...
<div className='searchBar' onKeyDown={(e) => searchKeyDown(e)}>
```

5.6.4. 활성화/비활성화

```
//컨트롤의 disabled 속성값에 boolean 으로 조건을 리턴하여 활성화/비활성화

let expuPtng = ""
let isExpired:boolean = true

<RadioButton ... disabled={expuPtng === 'R'}></RadioButton>
<Button ... disabled={isExpired}></Button>
```

5.6.5. 컨트롤 숨김/표시 처리

5.6.6. 컨트롤 속성 변경

```
//컨트롤에서 변경해야 할 속성 부분에 { } 괄호를 열고 값을 바인딩하여
//변수나 함수를 넣어줌으로써 직접적으로 속성을 바꿀 수 있다.

<InputText
  className={ 변수 or 값을 리턴하는 함수 }
```

```
placeholder={ 변수 or 값을 리턴하는 함수 }
value={ 변수 or 값을 리턴하는 함수 }
onChange={(e) => 함수() }
onKeyPress={ 함수() } />
```

5.6.7. 논리 && 연산자로 If를 인라인으로 표현

```
function Mailbox(props) {
  const unreadMessages = props.unreadMessages;
  return (
    <div>
      <h1>Hello!</h1>
      { unreadMessages.length > 0 &&
        <h2>
          You have {unreadMessages.length} unread messages.
        </h2>
      }
    </div>
  );
}
```

5.6.8. 조건부 연산자로 If-Else구문 인라인으로 표현

```
render() {
  const isAuthenticated = state.isLogin;
  return (
    <div>
      The user is <b>{ isAuthenticated ? 'currently' : 'not' }</b> login.
    </div>
    <div>
      { isAuthenticated
        ? <LogoutButton onClick={ handleLogoutClick } />
        : <LoginButton onClick={ handleLoginClick } />
      }
    </div>
  );
}
```

```
}

```

5.6.9. 컴포넌트 렌더링 방지

```
function WarningBanner(props) {
  if (!props.warn) {
    return null;
  }

  return (
    <div className="warning">
      Warning!
    </div>
  );
}
```

5.7. 서브미션 (API 연동)

5.7.1. 개요

Api 연동하여 화면을 만드는 작업에 대해 설명한다.

클라우드 포탈에서는 Redux-toolkit 의 Redux + thunk를 확장한 RTK Query 라이브러리를 사용하였다.

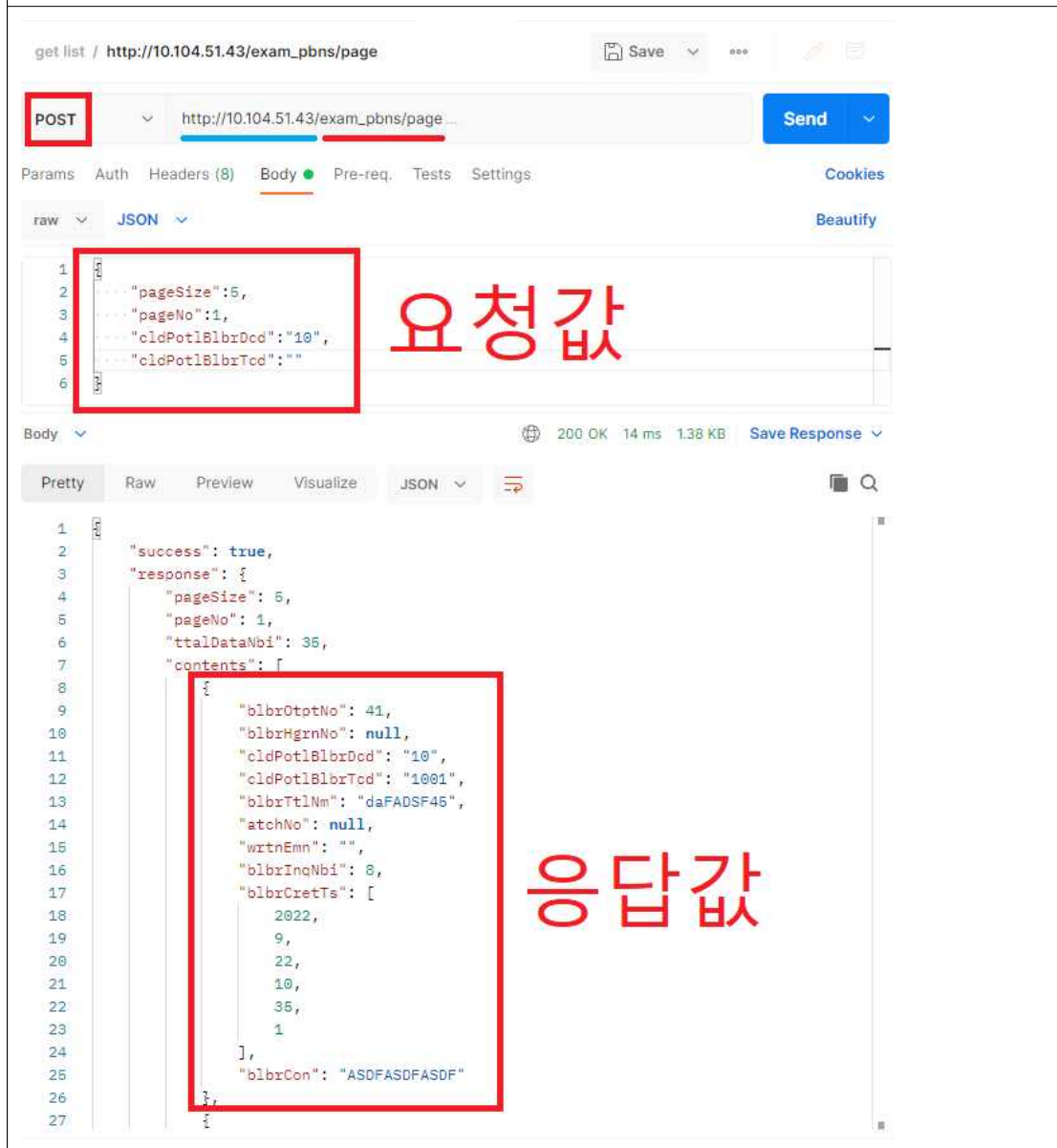
문서에서 다루지 않는 추가적인 RTK Query를 사용법은 웹페이지 참조.

<http://redux-toolkit.js.org/rtk-query/overview>

* 작업 전 전제조건

REST API 가 개발되어 있고, Postman 이나 Swagger 등으로 API를 테스트하여 결과값을 볼 수 있는 화면이 존재한다고 가정한다.

[그림6.7.1] 포스트맨 사용시



5.7.2. API 호출 URL 환경변수에 세팅

.env.loc (로컬 정보)
REACT_APP_TITLE =CLOUD_PORTAL REACT_APP_API_URL=http://10.104.51.49 REACT_APP_MODE=development
env 변수 참조한 부분 (src/app/core/services/base/api.service.ts)
<pre>let base_api = process.env.REACT_APP_API_URL // Create our baseQuery instance const baseQuery = fetchBaseQuery({ baseUrl: `\${base_api}`,</pre>

루트 디렉토리의 .env.loc 에 선언된 변수를

src/core/servies/base/api.service.ts 에 base_api에서 base url 로 참조하므로
시작하기 전에 불러올 api 주소로 환경변수를 바꿔준다.

*env 파일을 수정하면 터미널 창에서 Ctrl + C를 눌러서 로컬 서버 실행을 멈추고 다시
npm start를 해줘야 바뀐 환경변수대로 실행이 된다.

5.7.3. 데이터 인터페이스 생성

[그림6.7.1] 에서 응답값 부분의 json 영역을 복사하여
app/core/models 하위에 새로운 모델을 생성한다.

* 각 인터페이스의 파일명 및 인터페이스명은 프로젝트 내에서 유일한 명칭으로 작성한다.
한 프로젝트는 하나의 앱이기 때문에 파일명과 함수명이 같으면 충돌이 있을 수 있다.

post.ts
<pre>export interface Post = { blbrOtptNo?:string; //필수값이 아니고 없어도 있는 값이면 ?를 붙인다 cldPotlBlbrDcd:string; cldPotlBlbrTcd:string; blbrTtlNm:string; //제목 atchNo:string; blbrHgrnNo?:string; wrtnEmn:string; //작성자 blbrInqNbi?:number; blbrCon:string; //내용</pre>

post.ts
}

5.7.4. API 호출부 생성

src/app/core/services 하위에 새로운 api service를 생성한다.

* 파일 명명 규칙 : **.service.ts

전체 api 주소가

http://10.104.51.43/exam_pbns/page

http://10.104.51.43/exam_pbns/detail

http://10.104.51.43/exam_pbns/save

라면 suffixPath 에 exam_pbns를 넣어주고 나머지는 쿼리에 써준다.

* 각 API의 파일명 및 API 함수명은 프로젝트 내에서 유일한 명칭으로 작성한다. 한 프로젝트는 하나의 앱이기 때문에 파일명과 함수명이 같으면 충돌이 있을 수 있다.

Response.ts (공통 응답 데이터 모델)
<pre>export interface Response { error: boolean, success: boolean, response: any }</pre>
post.service.ts
<pre>import { Post, PostDetail } from '../models/post' //미리 정의한 interface 호출 import { Page } from '../models/page' import { Response } from '../models/response' import { api } from '../base/api.service' import { CommonResponse } from '../base/response.service' let suffixPath = 'exam_pbns' export const postApi = api.injectEndpoints({ endpoints: (build) => ({ //목록 조회 //query<ResultType, QueryArg> getPosts: build.query<Response, Partial<Post>>({ query: (body) => ({</pre>

post.service.ts

```

        url: `${suffixPath}/page`, //api 이름 선언
        method: 'POST', //GET, POST, PUT, DELETE 정의
        body,
    }},
    transformResponse: CommonResponse,
    //화면에서 에러가 나자마자 직접 관련 처리를 하고 싶으면
    //이 부분을 지우고 response.response를 받아 처리하면 됩니다
  }},
  //상세 조회
  getPostById: build.query<Response, Partial<PostDetail>>({
    query: (body) => ({
      url: `${suffixPath}/detail`,
      method: 'POST',
      body,
    }),
    transformResponse: CommonResponse,
  }},
  //등록
  //mutation<ResultType, QueryArg>
  addPost: build.mutation<Response, Partial<Page>>({
    query: (body) => ({
      url: `${suffixPath}/save`,
      method: 'POST',
      body,
    }),
    transformResponse: CommonResponse
  }},
  //수정
  updatePost: build.mutation<Response, Partial<Page>>({
    query: (body) => ({
      url: `${suffixPath}/save`,
      method: 'POST',
      body,
    }),
    transformResponse: CommonResponse,
  }},
  }},
  })

```

post.service.ts

```

/*
[export 명 규칙]

    HTTP 와 완전히 연동되는 것은 아님
    (예를 들어 데이터 조회시 POST 를 쓴다 해도 use***Query 라고 하면 됨)
    데이터를 조회하는 경우 use + 함수명(맨 앞글자 -> 대문자) + Query
    데이터를 변화시키는 경우 (등록, 수정, 삭제)
        use + 함수명(맨 앞글자 -> 대문자) + Mutation

    *mutation: 참조된 데이터가 변경되는 것
*/
export const {
    useGetPostsQuery,      //목록 조회
    useGetPostByIdQuery,   //상세 조회
    useAddPostMutation,    //등록
    useUpdatePostMutation, //수정
} = postApi

export const {
    endpoints: { getPosts, getPostById, addPost, updatePost },
} = postApi

```

5.7.5. 요청/응답 데이터 생성

페이지에서 api 호출

조회시

```

import {
    useGetPostsQuery
} from '../core/services/post.service'

const SampleList:React.FC = () => {

    //게시물 조회조건
    let param = {
        pageSize:10,
        pageGrpNbi : 5,

```

조회시

```

}

//목록조회 API 사용하여 데이터 조회
const { data:posts, isFetching, isLoading } = useGetPostsQuery( param )
//이 posts 는 데이터를 받아오는 일종의 변수명이다.
React.useEffect(()=> {
  if(posts) {
    let response:any = posts
    //조회한 데이터 관련 처리
  }
}, [posts]) //posts 값에 뭔가가 들어오거나 변경될 때 이 useEffect 실행됨

```

등록, 수정, 삭제시

```

import {
  useAddPostMutation,
} from '../core/services/post.service'

const SampleRegister:React.FC = () => {

  //등록 API 사용
  const [ addPost, { isLoading } ] = useAddPostMutation()

  const save = () => {
    console.log(`values ${values}`)
    addPost(values).unwrap()
      .then(( response:any ) => {
        if(response) {
          alert('게시물이 등록되었습니다.');
          goPage('/sample/list');
        }
      })
      .catch((error) => console.error('rejected', error))
  }

  return(<Button onClick={Save}>확인</Button>)
}

```

5.8. 화면 이동

5.8.1. 화면 이동 함수

React 에서는 `useNavigator()` hook을 사용하여 페이지 이동을 처리한다.

SamplePage.tsx

```
const SamplePage:React.FC = () => {
  const navigator = useNavigation()

  const goListPage = () => {
    navigator('/sample/list')
  }

  const cancel = () => {
    //뒤로가기
    navigator(-1)
  }

  return(
    <BasePage>
      <div className='detailFooterButton'>
        <Button onClick={cancel}>취소</Button>
        <Button onClick={goListPage}>목록으로 이동</Button>
      </div>
    </BasePage>)
}

export default SampleEdit;
```

```
//화면 템플릿에서는 link 사용
const url = '/sample/list'
...
<Link to={ url }>이동</Link>
```

공통적인 화면 이동 처리를 위해 커스텀 훅을 사용하면 매번 여러 유틸을 import 하는 번거로움을 줄일 수 있다.

SampleEdit.tsx

```
import useBasePage from '../shared/hooks/base-page.hook';

const SampleEdit:React.FC = () => {
  const { goPage, goBack } = useBasePage()

  const goListPage = () => {
    goPage('/sample/list')
  }
```

SampleEdit.tsx

```

    }
    const cancel = () => {
      //뒤로가기
      goBack();
    }
    return(
      <BasePage>
        <div className='detailFooterButton'>
          <Button onClick={cancel}>취소</Button>
          <Button onClick={goListPage}>목록으로 이동</Button>
        </div>
      </BasePage>)
    }
    export default SampleEdit;

```

5.8.2. 팝업

dialog 호출이 필요한 곳에서 confirmDialog 함수를 호출한다.

Confirm 팝업

```

import { confirmDialog } from 'primereact/confirmdialog';
import { Button } from 'primereact/button';

const SampleEdit:React.FC = () => {
  //제목이 있는 팝업
  const confirm1 = () => {
    confirmDialog({
      header: '제목 영역',
      message: '내용 영역',
      rejectLabel: '취소',
      acceptLabel: '확인',
      accept: () => { /* 확인 버튼 눌렀을 때 처리 */ },
      reject: () => { /* 취소 버튼 눌렀을 때 처리 */ }
    })
  }

  //제목이 없는 팝업
  const confirm2 = () => {

```


Confirm 팝업

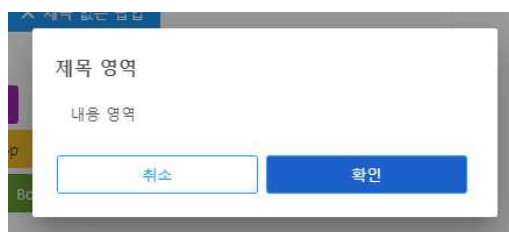
```

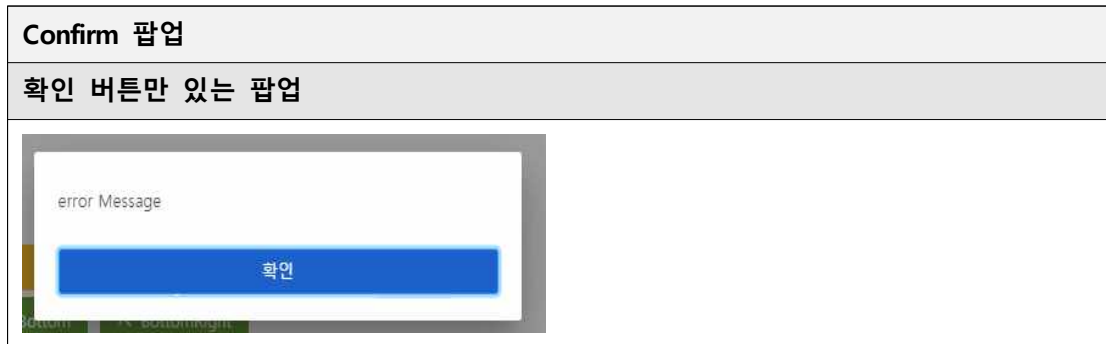
confirmDialog({
  message: '내용 영역',
  rejectLabel: '취소',
  acceptLabel: '확인',
  className: 'noHeader',
  accept: () => { /* 확인 버튼 눌렀을 때 처리 */ },
  reject: () => { /* 취소 버튼 눌렀을 때 처리 */ }
})
}

//확인 버튼만 있는 팝업
const confirm3 = () => {
  confirmDialog({
    message: 'error Message',
    className: 'noHeader oneButton',
    acceptLabel: '확인',
    accept: () => { /* 확인 버튼 눌렀을 때 처리 */ },
  })
}

return(
  <BasePage>
    <Button onClick={confirm1} icon="pi pi-check"
      label="제목 있는 팝업" className="mr-2" />
    <Button onClick={confirm2} icon="pi pi-times"
      label="제목 없는 팝업" />
    <Button onClick={confirm3} icon="pi pi-times"
      label="확인 버튼만 있는 팝업" />
  </BasePage>)
}
export default SampleEdit;

```

제목이 있는 팝업**제목이 없는 팝업**



5.9. 인증 처리

5.9.1. 개요

SSO 로그인, 로그아웃을 사용한다.

5.9.2. 로그인 Process

순서	페이지/파일명	처리 내용
1	메인화면	인증 token 이 있는지 체크 -> 없으면 자동 로그인
2	base-api.service.ts	환경변수에 지정된 REACT_APP_SSO_AUTH 호출 (http://***/sso)
3	Back-End	Back-End 단에서 SSO 로그인 처리하고 Front-End 의 Callback 페이지 호출하며 empid를 get 으로 전달 (보안을 위해 empid 는 암호화 처리)
4	Callback.tsx	empid 를 post 로 보내는 login api 호출 -> token, user 정보를 받아 localStorage 에 저장
5	api.service.ts	baseQuery에서 localStorage 의 token 과 empid를 api 공통 header 에 세팅. * token과 empid를 Api Request Header 에 넣어 보내지 않으면 Api가 정상적으로 동작하지 않는다.
6	메인화면	menu Api 호출 등 기타 Api 호출

* 세션 만료 등 기타 에러 예외 처리는 api.service.ts 의 baseQueryWithReauth를 참조

5.9.3. 로그아웃 Process

순서	페이지/파일명	처리 내용
1	Header.tsx	로그아웃 함수 호출

2	base-api.service.ts	localStorage, sessionStorage를 지우고 환경변수에 세팅된 REACT_APP_SSO_LOGOUT 호출 (http://***/logout)
---	---------------------	---

5.9.4. 클라우드 포탈 사용자, 관리자 특이사항

클라우드 포탈 사용자는 서버에서 token을 받아와서 인증을 처리한다.

클라우드 포탈 관리자는 클라우드 포탈 사용자에서만 접근 가능하다.

클라우드 포탈 관리자 접근 Process는 클라우드 포탈 사용자 화면 로그인 -> Header 의 프로필 정보 팝업의 '관리자' 클릭 -> 사용자의 token 과 empid 정보가 관리자의 localStorage 로 넘어간다 -> 해당 정보를 사용하여 관리자에서 로그인 처리한다.

세션 만료 전까지는 사용자에서 받아온 token 으로 관리자에 언제든지 재접속 가능하지만, 세션이 만료되면 (Back-end에서 판단) 클라우드 포탈 사용자 페이지에 되돌아가서 다시 접속하라는 페이지를 보여준다.

클라우드 포탈 사용자
Header.tsx <pre> ... { /* 프로필 이름 클릭시 나오는 패널 */ { profilePanelOpen && <ProfilePanel domRef={profileEl} logout={logoutFunc} > <li className='border-bottom'>{userName} [{emn}] { blngNm && <li className='border-bottom'>{blngNm} } {getIsAdmin() && <li className='border-bottom pl5 pt2 pb2'> <Button className='p-button-link' onClick={(e) => { // '관리자' 클릭시 index.html 로 이벤트 송신 const cstEvt = new CustomEvent('cstMsg'); document.dispatchEvent(cstEvt); closeProfile(); }} > 관리자 </Button> } } } } </pre>

}

index.html

...

<script>

//프로필 팝업에서 보내온 이벤트 수신

```
document.addEventListener('cstMsg', (e) => {
  postCrossDomainMessage();
})
```

//클라우드 포탈 관리자 페이지로 postMessage를 이용하여 정보 송신

```
function postCrossDomainMessage() {
  const ADMIN_URL = "%REACT_APP_ADMIN_URL%" + '?from=user';
  const msg = {
    empid: localStorage.getItem('empid'),
    access_token: localStorage.getItem('access_token'),
    user: localStorage.getItem('user'),
    iam: "%REACT_APP_IAM%"
  }
}
```

```
const win = window.open(ADMIN_URL, "_blank")
```

```
setTimeout(function() {
```

```
  win.postMessage(msg, ADMIN_URL)
```

```
}, 1000)
```

```
}
```

</script>

클라우드 포탈 관리자**index.tsx**

...

```
const hasToken = isValidToken()
```

```
const isAdmin = getIsAdmin()
```

```
const queryString = window.location.search;
```

```
const urlParams = new URLSearchParams(queryString)
```

```
const fromParam = urlParams.get('from');
```

// 사용자 -> 관리자로 접속한 경우 get parameter 로 판단

```
if(fromParam === 'user') {
```

```

root.render(loading());
window.addEventListener('message', showAdminMain, false );
}

...

function showAdminMain( event:any ) {
  // Event가 클라우드 포털 사용자에서 넘어온 이벤트인지 데이터를 열어서 확인한다
  let isEventFromUser:boolean = event?.data?.iam === ADMIN_KEY;

  if(isEventFromUser) {
    if(!event.origin.includes(USER_DOMAIN)) return;

    // 클라우드 포털 사용자에서 넘어왔을 경우 정보를 localStorage 에 저장
    if(event.origin.includes(USER_DOMAIN)) {
      let data = event.data;
      for(let key in data) {
        localStorage.setItem(key, data[key])
      }
      //localStorage 에 필요한 정보 저장 후 화면을 렌더링
      root.render(rendering());
    }
  }
}

```

5.10. 권한

5.10.1. 개요

권한에 따라 사용자/관리자 페이지 접근을 제어하는 작업에 대해 설명한다.
예시로 클라우드 포털에서는 아래와 같은 3가지 권한을 이용한다.

코드	권한	설명
10	사용자	관리자용 페이지 접근 불가
11	설문 관리자	사용자 설문관리 페이지 접근은 가능하되 본인이 등록한 설문만 확인, 수정 가능
01	관리자	모든 페이지 접근, 확인, 수정 가능

5.10.2. 권한을 체크하는 공통 함수

src/app/core/services/base/base-api.service.ts

```
const getIsAdmin = ():boolean => {
  let userData = JSON.parse(getUserInfo())
  let isAdmin = false;

  if(userData && userData.cldPotlAthrList) {
    isAdmin = userData.cldPotlAthrList.includes(
      commonCode.CLD_POTL_ATHR_DCD_ADM)
  }
  return isAdmin;
}
```

5.10.3. 권한별 화면 분기

권한이 없을 경우 화면에서 특정 UI 가 안보이게 하는 처리

예시)

```
return (<BasePage>
  {
    getIsAdmin() ? <Button label='관리자 화면' />
  }
</BasePage>)
```

5.10.4. 권한별 화면 접근 처리

router에서 권한에 따라 분기 처리하여, url을 직접 치고 들어와도 접근할 수 없도록 설정한다.

클라우드 포탈 사용자 icp-clp-front-user

1. 라우팅 분기 처리 src/app/routes/portal-routes.js

- * 설문 관리자: 설문 관리 화면 접근 가능
- * 관리자: 전부 접근 가능

/*

* 권한 관련 속성 추가

설문관리 : 설문관리자 또는 관리자만 접근 가능

이벤트 : 관리자 또는 행내 직원만 접근 가능

특정 조건의 권한만 접근 가능한 경우

element 속성에 아래 조건문을 추가해 줘야 함 (예: '설문 관리' 참조)

element: isAuthenticated.hasToken && !isAuthenticated.isAdmin ?

<Navigate to='/auth/not-admin' /> : <페이지 컴포넌트 />

*/

const iff = (condition, then, otherwise) => condition? then : otherwise;

const portalRoutes = (isAuthenticated) => [

{

path: '/',

element: iff(isAuthenticated.hasToken ,

<Navigate to='/man' /> , <Navigate to='/auth/logout' />),

},

{

path: '/',

element: iff(isAuthenticated.hasToken ,

<FullLayout /> , <Navigate to='/auth/logout' />),

children: [

{ path: 'man',

name: '메인',

element: <CLPMANM00100 />,

},

{

path: 'sgs',

name: '설문관리',

children: [

{

path: 'sur',

name:'설문관리',

children : [

{path: 'list', name:'설문 관리',

element:

iff(isAuthenticated.hasToken && !(isAuthenticated.isAdmin || isAuthenticated.isSurAdmin) ,

<Navigate to='/auth/not-admin' /> , <CLPSURM93310 />)),

]

},

],

},

...

클라우드 포탈 관리자 : icp-clp-front-admin

1. 라우팅 분기 처리 : src/app/routes/admin-routes.js

* 관리자 화면은 관리자만 접근 가능

```

const getAccessView = (isAuthenticated, type) => {
  let view;

  if(!isAuthenticated.hasToken) {
    //인증 토큰이 없으면
    view = <Navigate to='/auth/logout' /> //토큰을 받아온다
  }
  else if(isAuthenticated.hasToken && !isAuthenticated.isAdmin) {
    //토큰이 있는데 관리자가 아니면 관리자 아님 페이지로 이동
    view = <Navigate to='/auth/not-admin' />
  }
  else if(isAuthenticated.hasToken && isAuthenticated.isAdmin) {
    //토큰 있고 관리자이면 메인 or <FullLayout /> 으로 이동
    if(type === '1') {
      view = <Navigate to='/man' />
    }
    else {
      view = <FullLayout />
    }
  }
  else {
    view = <Navigate to='/auth/logout' />
  }

  return view;
}

const adminRoutes = (isAuthenticated) => [
  /* [인증]
    1. 토큰을 가지고 있지 않으면 (!isAuthenticated.hasToken)
    2. logout페이지로 가서 ssoLogin 자동로그인을 호출한다.
    3. ssoLogin 해서 토큰이 있는데 admin 권한이 없다면 auth/not-admin(권한이 없습니다)
       페이지로 간다.
    4. 토큰이 있고 관리자이면 main 으로 이동한다.
  */
  {

```



```

    path: '/',
    element : getAccessView(isAuthenticated, '1'),
  },
  {
    path: '/',
    element: getAccessView(isAuthenticated, '2')
    children: [
      {
        path: 'ui',
        ...

```

2. URL 직접 접근 시 권한별 접근 제어 : src/index.tsx

```

/**
1. 사용자 권한 체크
2. 관리자가 아니면 root.rendering을 막고
   클라우드 포탈에서 관리자 권한 신청하라는 안내 페이지를 띄운다
**/

...

const hasToken = isValidToken()
const isAdmin = getIsAdmin()
const queryString = window.location.search;
const urlParams = new URLSearchParams(queryString)
const fromParam = urlParams.get('from');

// 사용자 -> 관리자로 접속한 경우 get parameter 로 판단
if(fromParam === 'user') {
  root.render(loading());
  window.addEventListener('message', showAdminMain, false );
}
// 관리자 내에서 화면 렌더링 하는 경우 (url 직접 접근 등)
else {
  //토큰이 있으면 렌더링
  if(hasToken && isAdmin) {
    root.render(rendering());
  }
  //토큰이 있고 관리자가 아닌 경우 관리자 권한 신청 안내 페이지로 이동
  else if(hasToken && !isAdmin) {
    root.render(<Logout />);

```

```
}  
else {  
  root.render(<Logout />);  
  
  // 토큰이 없는 경우에는 사용자 페이지에서 받아온다  
  window.addEventListener('message', showAdminMain, false );  
}  
}
```

5.11. 유의사항

5.11.1. 라이브러리 추가 설치 시 주의사항

SSO 최신 버전이 아직 Beta 버전이라서 만일을 대비하여 Edge 브라우저의 IE11 모드를 지원해야 하므로 IE11 브라우저에서 작동시 문제가 없는지 테스트 하고 추가 설치 필요.

5.11.2. 성능

array.map, for 등 반복문을 리액트 UI에서 사용할 경우 성능상의 이점을 위하여 key 값을 꼭 넣어준다.

(예를 들어 리스트 순서를 바꿀 때 key 값이 없으면 전체 리스트를 렌더링, key 값이 있으면 이동하는 해당 리스트만 다시 그림)

6. 예제 (게시판 생성, 조회, 수정)

Summary

페이지 생성 (퍼블리셔)
src > app > pages > 페이지 생성 src > app > routes > routes.js 에 페이지 링크 추가
api 바인딩 (개발자)
src > app > core > models > 데이터 인터페이스 파일 생성 src > app > core > services > CRUD api 정의 파일 생성 src > app > pages > 에 정의한 페이지에 api 호출

6.1. 페이지 생성

6.1.1. 페이지 생성

일반 페이지 컴포넌트 생성

src/app/pages/[페이지폴더명]/[페이지이름].tsx 파일 생성

SamplePage.tsx

//사용하는 공통 라이브러리 등 선언부

import * as React from 'react';

import useBasePage from '../shared/hooks/base-page.hook';

const SampleEdit:React.FC = () => {

//페이지 공통 hook 중 사용할 함수, 변수를 임포트.

//이렇게 쓰면 goPage('/notice/list') 등 함수를 바로 사용 가능

const { paramId, goPage, goBack, commonCode, BasePage } = useBasePage()

const [detail, setDetail] = React.useState(null)

//리액트의 hook 은 컴포넌트 상단에 선언 (useState, useEffect 등)

//hook 은 일반 함수 내에서 사용 불가

//페이지 로딩되고 초기에 한 번만 실행되는 hook

React.useEffect(()=> {

//실행할 함수 작성

}, []) //<-여기를 [] 빈 배열로 놓아야 초기 한 번만 실행됨.

//아무 값도 없으면 무한 호출되므로 주의

React.useEffect(()=> {

SamplePage.tsx

```

    //실행할 함수 작성
    }, [ detail ]) //<- 이 배열 안에 변수가 들어있으면, 이 값에 변동이 있을 때마다
    //해당 useEffect hook이 실행된다

    //return 부분에 렌더링될 UI 작성
    //UI 요소는 단일한 컴포넌트가 되도록 <></> 또는 <div></div> 안에 묶어줘야 함
    //클라우드 포탈의 경우 페이지들은 공통 컴포넌트인 <BasePage> 안에 작성
    return(
      <BasePage>
        <h1>Hello, world! </h1>
      </BasePage>)
  }
  export default SampleEdit;

```

6.1.2. route 에 페이지 추가

src/app/routes/routes.js 에 생성한 페이지를 추가한다.

sample-routes.js

```

import { lazy } from 'react';
import { Navigate } from 'react-router-dom';
import Loadable from '../shared/layouts/loader/Loadable';

/***** Pages *****/
//페이지 컴포넌트명, 경로 추가
const SampleList = Loadable(lazy(async () => await import('../pages/sample/SampleList')));
const SampleDetail = Loadable(lazy(async () => await import('../pages/sample/SampleDetail')));
const SampleRegister = Loadable(lazy(async () =>
  await import('../pages/sample/SampleRegister')));
const SampleEdit = Loadable(lazy(async () => await import('../pages/sample/SampleEdit')));

/*****Routes*****/
const Routes = (isLoggedIn) => [
  {
    path: '/sample',
    element: <FullLayout />,
    children: [
      //목록화면

```

sample-routes.js

```

    { path: 'list', name: 'sample list', exact: true, element: <SampleList /> },
    //상세화면 (주소창에 [도메인]/sample/detail/1 이런식으로 id 로 접근 가능)
    { path: 'detail/:id', name: 'sample detail', exact: true, element: <SampleDetail /> },
    { path: 'edit/:id', name: 'sample SampleEdit', exact: true, element: <SampleEdit /> },
  ],
},
...
];

export default Routes;

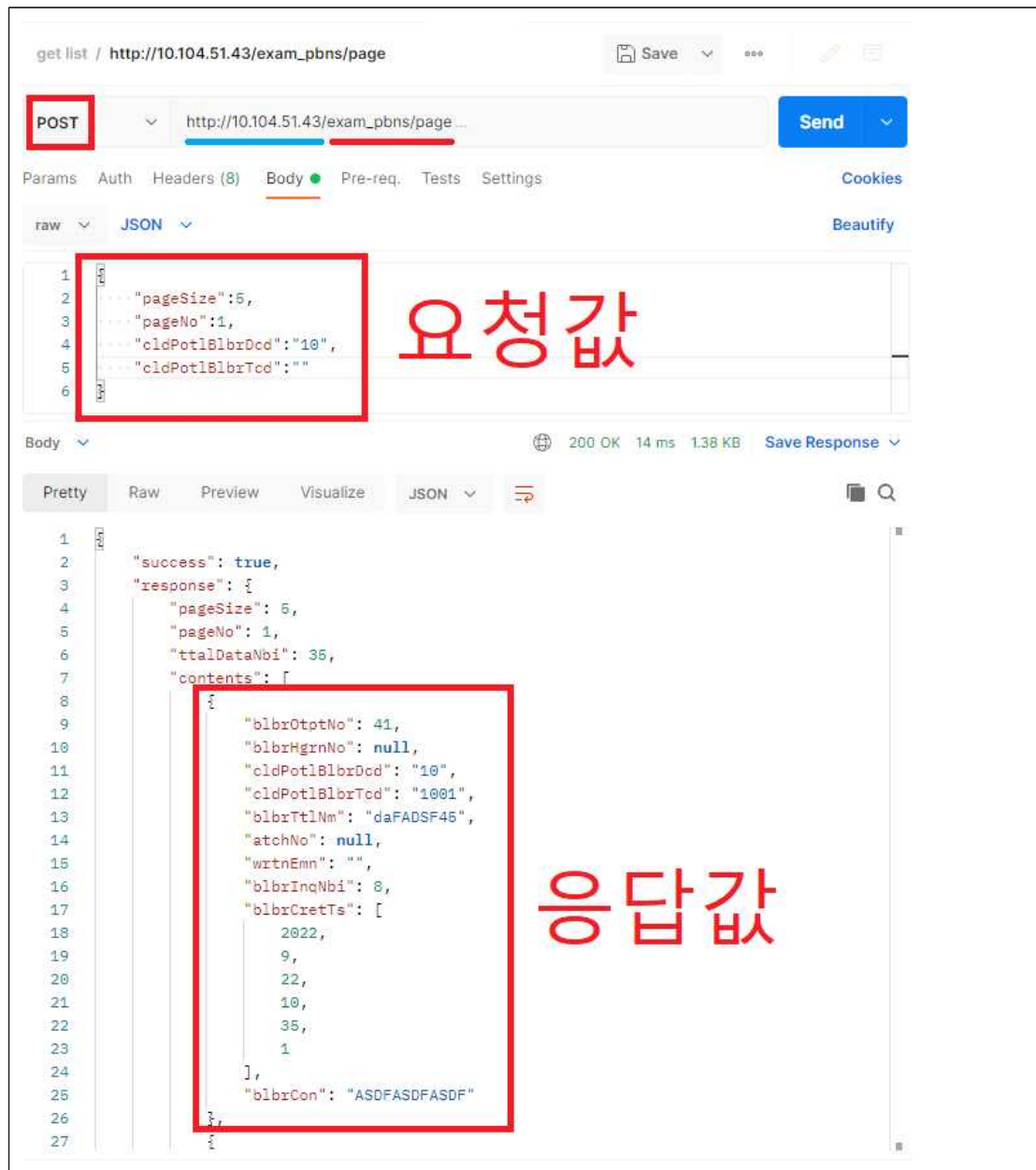
```

6.2. API 바인딩

* 작업 전 전제조건

- REST API 가 개발되어 있고, Postman 이나 Swagger 등으로 API를 테스트하여 결과값을 볼 수 있는 화면이 존재한다고 가정한다.
- 퍼블리셔가 화면을 만들어서 routing 까지 구성이 되어있다.

[그림 7.2] 포스트맨 사용시



6.2.1. API 호출 URL 환경변수에 세팅

.env.loc (로컬 정보)

```

REACT_APP_TITLE =CLOUD_PORTAL
REACT_APP_API_URL=http://10.104.51.49
REACT_APP_MODE=development

```

루트 디렉토리의 .env.loc 에 선언된 변수를 base_api에서 base url 로 참조하므로 시작하기 전에 **블러울 api 주소로 환경변수를 바꿔준다.**

*** env 파일을 수정하면 터미널 창에서 Ctrl + C를 눌러 로컬 서버 실행을 멈추고 다시 npm start를 해줘야 바뀐 환경변수대로 실행이 된다.**

6.2.2. 데이터 인터페이스 생성

[그림6.7.1] 에서 응답값 부분의 json 영역을 복사하여
app/core/models 하위에 새로운 모델을 생성한다.

post.ts

```
export interface Post = {
  blbrOtpNo?:string; //필수값이 아니고 없을 수도 있는 값이면 ?를 붙인다
  cldPotlBlbrDcd:string;
  cldPotlBlbrTcd:string;
  blbrTtlNm:string; //제목
  atchNo:string;
  blbrHgrnNo?:string;
  wrtnEmn:string; //작성자
  blbrInqNbi?:number;
  blbrCon:string; //내용
}
```

6.2.3. API 호출부 생성

src/app/core/service 하위에 새로운 api service를 생성한다.

* 파일 명명 규칙 : **.service.ts

전체 api 주소가 http://10.104.51.43/exam_pbns/page 라면
suffixPath 에 exam_pbns를 넣어주고 나머지는 쿼리에 써준다.

post.service.ts

```
import { Post, PostDetail } from '../models/post' //미리 정의한 interface 호출
import { Page } from '../models/page'
import { Response } from '../models/response'
import { api } from '../base/api.service'
import { CommonResponse } from '../base/response.service'

let suffixPath = 'exam_pbns'

export const postApi = api.injectEndpoints({
  endpoints: (build) => ({
    //목록 조회
    getPosts: build.query<Response, Partial<Post>>({
      query: (body) => ({
        url: `${suffixPath}/page`, //api 이름 선언
        method: 'POST', //GET, POST, PUT, DELETE 정의
      })
    })
  })
})
```

post.service.ts

```

        body,
      }),
      transformResponse: CommonResponse,
    }),
    //상세 조회
    getPostById: build.query<Response, Partial<PostDetail>> >({
      query: (body) => ({
        url: `${suffixPath}/detail`,
        method: 'POST',
        body,
      }),
      transformResponse: CommonResponse,
    }),
    //등록
    addPost: build.mutation<Response, Partial<Page>> >({
      query: (body) => ({
        url: `${suffixPath}/save`,
        method: 'POST',
        body,
      }),
      transformResponse: CommonResponse
    }),
    //수정
    updatePost: build.mutation<Response, Partial<Page>> >({
      query: (body) => ({
        url: `${suffixPath}/save`,
        method: 'POST',
        body,
      }),
      transformResponse: CommonResponse,
    }),
  })),
})

/*
[export 명 규칙]

```

HTTP 와 완전히 연동되는 것은 아님
(예를 들어 데이터 조회시 POST 를 쓴다 해도 use***Query 라고 하면 됨)

post.service.ts

데이터를 조회하는 경우 use + 함수명(맨 앞글자 -> 대문자) + Query
 데이터를 변화시키는 경우 (등록, 수정, 삭제)

use + 함수명(맨 앞글자 -> 대문자) + Mutation

*mutation: 참조된 데이터가 변경되는 것

*/

```
export const {
  useGetPostsQuery,      //목록 조회
  useGetPostByIdQuery,   //상세 조회
  useAddPostMutation,    //등록
  useUpdatePostMutation, //수정
} = postApi

export const {
  endpoints: { getPosts, getPostById, addPost, updatePost },
} = postApi
```

여기서 만든 api를 화면에서 호출하여 작업한다.

6.2.4. 목록 화면**SampleList.tsx**

```
import * as React from 'react';
import { Button, Column, DataTable } from 'primereact';
import {
  useGetPostsQuery,
} from '../core/services/post.service'
import useBasePage from '../shared/hooks/base-page.hook';
import { paginator } from '../shared/utlis/base-table';
import { TableHead } from '../core/models/table-head';

//테이블 헤더 내용 정의 (퍼블리셔가 1차적으로 작성)
const headCells: TableHead[] = [
  {
    field: 'id', //참조하는 값
    header: 'ID', //테이블 헤더에 표출되는 제목
    sortable: false, //sorting 관련내용 추후 업데이트
```

SampleList.tsx

```

    style: { width: '7%' } //테이블 너비
  },
  {
    field: 'subject',
    header: '제목',
    sortable: false,
    style: { width: '25%' }
  },
  {
    field: 'file',
    header: '첨부파일',
    sortable: false,
    style: { width: '10%' }
  },
  {
    field: 'hit',
    header: '조회수',
    sortable: false,
    style: { width: '10%' }
  },
  {
    field: 'date',
    header: '등록일',
    sortable: false,
    style: { width: '15%' }
  },
];

const SampleList:React.FC = () => {
  const { goPage, BasePage } = useBasePage()
  const [boardRow, setBoardRows] = React.useState([])
  const [first, setFist] = React.useState(0); //읽어올 데이터 index 시작점
  const [rows, setRows] = React.useState(10); //표시할 행 개수

  const handleCustomPage = (event) => {
    setFist(event.first);
    setRows(event.rows);
  }
}

```

SampleList.tsx

```

//게시물 조회조건
let param = {
  pageSize: 10,
  pageNo: 1,
  pageGrpInqYn : 'Y',
  pageGrpNbi : 5,
  cldPotlBlbrDcd:'10',
  cldPotlBlbrTcd:''
}
/*
  pageSize : 페이지당 보여지는 게시물 건수
  pageNo : 현재 조회할 페이지 번호
  pageGrpInqYn : 페이지그룹 단위로 Back-end 로 조회할 지 여부, Y이면 이경우 50
  건 단위로 Back-end에서 조회해줌, 아니면 그냥 10건으로 조회해줌
  pageGrpNbi : 페이지번호 그룹(하단)
*/

//목록조회 API 사용하여 데이터 조회
const { data:posts, isFetching, isLoading } = useGetPostsQuery( param )

//페이지 로딩되고 초기에 실행되는 hook
React.useEffect( ()=> {

}, [] ) //<-여기를 [] 빈 배열로 놓아야 초기 한 번만 실행됨

React.useEffect( ()=> {

  if(posts) {
    let response:any = posts
    let boardData = response.contents

    //rows 데이터 배열 초기화
    setBoardRows([])

    //데이터 읽어와서 필요한 데이터만 골라서 테이블 배열에 추가
    boardData.map(({blbrOtptNo, blbrTtlNm, blbrInqNbi, blbrCretTs, wrtnEmn}) =>
    {
      let obj = {
        id: blbrOtptNo, //번호

```

SampleList.tsx

```

        subject: blbrTtlNm,           //제목
        file: "",                     //첨부파일
        hit: blbrInqNbi,              //조회수
        date: blbrCretTs.join('-')    //날짜
    };

    //rows 데이터 업데이트
    setBoardRows((prev) => (
        [...prev, obj]
    ))
    })
    }

}, [posts]) //posts 값에 뭔가가 들어오거나 변경될 때 실행됨

const handleRowClick = (e:any) => {
    let id = e.data.id;
    goPage(`/sample/detail/${id}`);
}

return (
    <BasePage>

        <div className='heading'>
            <h1>게시판 관리 (샘플)</h1>
        </div>

        <div style={{display: 'flex', justifyContent: 'justify-end', marginBottom: 10}}>
            <Button style={{width:100, justifyContent: 'center'}}
                onClick={()=>{goPage("/sample/register")}}>등록</Button>
        </div>

        <DataTable
            value={boardRow}
            paginator
            paginatorTemplate={paginator}
            first={first}
            rows={rows}
            onPage={handleCustomPage}

```

SampleList.tsx

```

    responsiveLayout='scroll'
    onClick={handleRowClick}
  >
    {headCells.map(({field, header, sortable, style}, index) => (
      <Column key={field} field={field} header={header}
        sortable={sortable} style={style}> </Column>
    ))}
  </DataTable>
</BasePage>
);
}
export default SampleList;

```

6.2.5. 상세 화면**SampleDetail.tsx**

```

import * as React from 'react';
import { useState, useEffect } from 'react';
import { postApi, useGetPostByIdQuery } from '../core/services/post.service';
import { Button } from 'primereact';
import useBasePage from '../shared/hooks/base-page.hook';

const SampleDetail:React.FC = () => {
  const { paramId, goPage, BasePage } = useBasePage()

  //게시물 조회조건
  let param = {
    "blbrOtpNo": paramId
  }

  //useQuery 는 object 로 받고, mutation 은 [] 배열로 받는다.
  //내용조회 API 사용하여 데이터 불러옴
  //data 를 post 라는 변수에 받아오겠다는 선언
  const {data:post, error} = useGetPostByIdQuery(param)

  const [detail, setDetail] = useState(null)

  //페이지 로딩되고 초기에 실행되는 hook

```

SampleDetail.tsx

```

useEffect(()=> {

    //게시판 내용 조회
    let boardData = post
    console.log('boardData =>', boardData)
    setDetail(boardData)

}, [post]) //post 에 값이 들어오거나 변경되면 이 useEffect 가 실행됨

const goEditPage = () => {
    goPage(`/sample/edit/${paramId}`)
}

const goListPage = () => {
    goPage(`/sample/list`)
}

if(!detail) {
    return (
        <h1>Loading</h1>
    )
}

return(
    <BasePage>
        <div className='heading'>
            <h1>게시판 조회 (샘플)</h1>
        </div>
        <div className='mb'>
            <p>등록자 정보</p>
            <table className='table'>
                <caption>등록자 정보</caption>
                <tbody>
                    <tr>
                        <th>등록자</th>
                        <td>s12345 {detail.wrtnEmn}</td>

                        <th>등록일시</th>
                        <td>{detail.blbrCretTs.join('.')}</td>
                    </tr>
                </tbody>
            </table>
        </div>
    </BasePage>
)

```

SampleDetail.tsx

```

        </tr>
      </tbody>
    </table>
  </div>
  <div>
    <p>등록 내용</p>
    <table className='table'>
      <caption>등록내용</caption>
      <tbody>
        <tr>
          <th>구분</th>
          <td>공지사항</td>
        </tr>
        <tr>
          <th>제목</th>
          <td>{detail.blbrTtlNm}</td>
        </tr>
        <tr>
          <th>내용</th>
          <td>{detail.blbrCon}</td>
        </tr>
      </tbody>
    </table>
  </div>

  <div className='detailFooterButton'>
    <Button style={{width:100, justifyContent: 'center'}}
      onClick={goListPage} className="mr10">목록</Button>
    <Button style={{width:100, justifyContent: 'center'}}
      onClick={goEditPage} >수정</Button>
  </div>
</BasePage>)
}
export default SampleDetail;

```

6.2.6. 등록 화면**SampleRegister.tsx**

SampleRegister.tsx

```

import * as React from 'react';
import { useState } from 'react';
import {
  useAddPostMutation,    //등록
} from '../core/services/post.service'
import { Post } from '../core/models/post'
import useBasePage from '../shared/hooks/base-page.hook';
import { Button, Dropdown, InputText, InputTextarea } from 'primereact';

const SampleRegister:React.FC = () => {
  const { goPage, goBack, commonCode, BasePage } = useBasePage()

  //등록
  const [addPost, { isLoading }] = useAddPostMutation()

  const [selectedCity1, setSelectedCity1] = React.useState<any>(null);

  const [values, setValues] = useState<Post>({
    cldPotlBlbrDcd:'10',//"/"//
    cldPotlBlbrTcd:'1001',
    blbrTtlNm:"", //제목
    atchNo:"",
    blbrHgrnNo:"",
    wrtnEmn:"", //작성자
    blbrInqNbi:0,
    blbrCon:"", //내용
  });

  const handleChange = (prop: keyof Post) => (event) => {
    console.log(event.target.value)
    setValues({ ...values, [prop]: event.target.value });
  };

  const cancel = () => {

    //뒤로가기
    goBack()
  }
}

```


SampleRegister.tsx

```

const save = () => {
  console.log(`values ${values}`)

  addPost(values).unwrap()
    .then((payload) => {
      console.log('payload fulfilled =>',payload)

      if(payload.success) {
        alert('게시물이 등록되었습니다.');
        goPage('/sample/list');
      }
    }).catch((error) => console.error('rejected', error))
}

return(
  <BasePage>
    <div className='heading'>
      <h1>게시판 등록 (샘플)</h1>
    </div>
    <div className='mb'>
      <p className="mb10">등록자 정보</p>
      <table className='table'>
        <caption>등록자 정보</caption>
        <tbody>
          <tr>
            <th>등록자</th>
            <td>a12345</td>

            <th>등록일시</th>
            <td>2023.03.02. 15:00:00</td>
          </tr>
        </tbody>
      </table>
    </div>
    <div>
      <p className="mb10">등록 내용</p>
      <table className='table'>
        <caption>등록내용</caption>
        <tbody>

```

SampleRegister.tsx

```

    <tr>
      <th>구분</th>
      <td>
        <Dropdown
          value={values.cldPotlBlbrDcd || ''}
          options={commonCode.CLD_POTL_BLBR_DCD}
          onChange={handleChange('cldPotlBlbrDcd')}
          optionLabel="name"
          placeholder="Select a City" />
      </td>
    </tr>
    <tr>
      <th>제목</th>
      <td>
        <InputText
          value={values.blbrTtlNm}
          onChange={handleChange('blbrTtlNm')} />
      </td>
    </tr>
    <tr>
      <th>내용</th>
      <td>
        <InputTextarea
          rows={5}
          cols={30}
          value={values.blbrCon}
          style={{ width: '100%' }}
          onChange={handleChange('blbrCon')} />
      </td>
    </tr>
    <tr>
      <th>첨부파일</th>
      <td>
        구현 예정 :)
      </td>
    </tr>
  </tbody>
</table>
</div>

```

SampleRegister.tsx

```

    <div className='detailFooterButton'>
      <Button style={{width:100, justifyContent: 'center'}}
        onClick={cancel} className="mr10">취소</Button>
      <Button style={{width:100, justifyContent: 'center'}}
        onClick={save}>확인</Button>
    </div>
  </BasePage>)
}
export default SampleRegister;

```

6.2.7. 수정 화면**SampleEdit.tsx**

```

import * as React from 'react';
import {
  useGetPostByIdQuery,    //상세조회
  useUpdatePostMutation,  //수정
} from '../core/services/post.service'
import { Post } from '../core/models/post'
import useBasePage from '../shared/hooks/base-page.hook';
import { Button, InputText, InputTextarea } from 'primereact';

const SampleEdit:React.FC = () => {
  const { paramId, goPage, goBack, commonCode, BasePage } = useBasePage()

  const goListPage = () => {
    goPage('/sample/list')
  }

  //내용조회 API 사용하여 데이터 불러옴
  //게시물 조회조건
  let param = {
    "blbrOtpNo": paramId
  }

  const {data:post, error} = useGetPostByIdQuery(param)

```

SampleEdit.tsx

```

const [startLoading, setStartLoading] = React.useState(false)
const [updatePost, { isLoading: isUpdating }] = useUpdatePostMutation()
const [detail, setDetail] = React.useState(null)

const [values, setValues] = React.useState<Post>({
  blbrOtpNo: "",
  cldPotlBlbrDcd:"10",
  cldPotlBlbrTcd:"1001",
  blbrTtlNm:"", //제목
  atchNo:"",
  wrtnEmn:"",
  blbrCon:"", //내용
});

React.useEffect(()=> {

  //게시판 내용 조회
  let boardData:any = post
  console.log('boardData =>', boardData)

  setDetail(boardData)
  setValues(boardData)

}, [post]) //post 에 값이 들어오거나 변경되면 이 useEffect 가 실행됨

const Cancel = () => {

  //뒤로가기
  goBack();

  //버튼 로딩바 안보이게 하기
  setStartLoading(false);
}

const Save = () => {

  //버튼 로딩바 보이게 하기
  setStartLoading(true);
}

```

SampleEdit.tsx

```

updatePost(values).unwrap()
.then(( response:any ) => {
    console.log('updatePost response =>',response)

    if(response) {
        //버튼 로딩바 안보이게 하기
        setStartLoading(false);

        alert('게시물이 수정되었습니다.');
```

goPage(`/sample/detail/\${paramId}`)

```

    }
}).catch((error) => console.error('rejected', error))
}

const handleChange = (prop: keyof Post) => (event) => {
    setValues({ ...values, [prop]: event.target.value });
};

if(!detail) {
    return (
        <h1>Loading</h1>
    )
}

return(
    <BasePage>
        <div className='heading'>
            <h1>게시판 수정 (샘플)</h1>
        </div>
        <div className='mb'>
            <p>등록자 정보</p>
            <table className='table'>
                <caption>등록자 정보</caption>
                <tbody>
                    <tr>
                        <th>등록자</th>
                        <td>s12345 {detail.wrtnEmn}</td>

```

SampleEdit.tsx

```

        <th>등록일시</th>
        <td>{detail.blbrCretTs.join('.')}</td>
    </tr>
</tbody>
</table>
</div>
<div>
    <p>등록 내용</p>
    <table className='table'>
        <caption>등록내용</caption>
        <tbody>
            <tr>
                <th>구분</th>
                <td>공지사항</td>
            </tr>
            <tr>
                <th>제목</th>
                <td>
                    <InputText
                        value={values.blbrTtlNm}
                        onChange={handleChange('blbrTtlNm')}
                    />
                </td>
            </tr>
            <tr>
                <th>내용</th>
                <td>
                    <InputTextarea
                        rows={5}
                        value={values.blbrCon}
                        onChange={handleChange('blbrCon')}
                    />
                </td>
            </tr>
        </tbody>
    </table>
</div>

<div className='detailFooterButton'>

```

SampleEdit.tsx

```
        <Button style={{width:100, justifyContent: 'center'}}
            onClick={Cancel} className="mr10">취소</Button>
        <Button style={{width:100, justifyContent: 'center'}}
            onClick={Save} loading={startLoading}>확인</Button>
    </div>
</BasePage>)
}
export default SampleEdit;
```

7. 배포

7.1. npm run prd

터미널에 `npm run build` 명령어를 치면 `build` 폴더 하위에 결과물 페이지가 생성된다. `npm start` 할 때 나타나지 않은 에러가 보일 수 있으므로 `git push` 하기 전에 한 번씩 확인하면 좋다. (필수는 아님)

7.2. index.html 흰 화면 방지

배포시 흰 화면 방지를 위해 `package.json` 에 다음과 같은 설정을 추가한다.

package.json

```
{
  "name": "icp-clp-front-user",
  "version": "0.0.0",
  "private": true,
  "homepage": "/",
  "dependencies": {
    "@amcharts/amcharts5": "5.2.29",
    "@lottiefiles/react-lottie-player": "3.4.7",
    ...
  }
}
```

8. 기타

8.1. 운영에서 console.log 방지

index.tsx

```
if(process.env.NODE_ENV === 'production') {
  console.log = () => {}
  console.error = () => {}
  console.debug = () => {}
}
```

8.2. 운영에서 오른쪽 마우스 클릭, F12 키 방지

index.tsx

```
if(process.env.NODE_ENV === 'production') {
  window.onload = function () {
    document.addEventListener("contextmenu", function (e:any) {
```

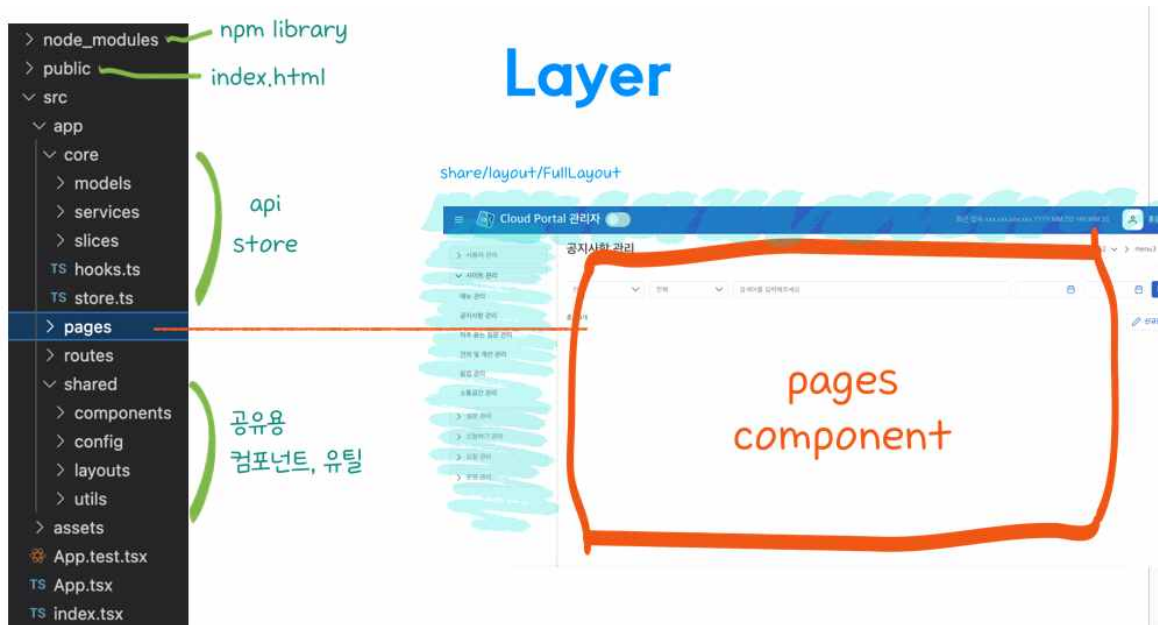

index.tsx

```

        e.preventDefault();
    }, false);
    document.addEventListener("keydown", function (e:any) {
        //document.onkeydown = function(e) {
        // "I" key
        if (e.ctrlKey && e.shiftKey && e.keyCode == 73) {
            disabledEvent(e);
        }
        // "J" key
        if (e.ctrlKey && e.shiftKey && e.keyCode == 74) {
            disabledEvent(e);
        }
        // "S" key + macOS
        if (e.keyCode == 83 && (navigator.platform.match("Mac") ? e.metaKey :
e.ctrlKey)) {
            disabledEvent(e);
        }
        // "U" key
        if (e.ctrlKey && e.keyCode == 85) {
            disabledEvent(e);
        }
        // "F12" key
        if (e.keyCode == 123) {
            disabledEvent(e);
        }
    }, false);
    function disabledEvent(e:any) {
        if (e.stopPropagation) {
            e.stopPropagation();
        } else if (window.event) {
            window.event.cancelBubble = true;
        }
        e.preventDefault();
        return false;
    }
}
}
}

```

9. [별첨1.] 화면 구조 ([별첨1] 참조)



index.tsx

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

App.tsx

```
import portalRoutes from './app/routes/portal-routes';

function App() {
  const routing = useRoutes(portalRoutes);

  return (
    <>
      <Suspense fallback={<Loader />}>
        {routing}
      </Suspense>
    </>
  );
}
```

portal-router.js

```
const adminRoutes = [
  {
    path: '/',
    element: <FullLayout />,
    children: [
      {
        path: 'notice',
        name: 'notice',
        children: [
          {path: 'list', element: <NoticeList />},
          {path: 'register', element: <NoticeRegister />},
          {path: 'detail/:id', element: <NoticeDetail />},
          {path: 'edit/:id', element: <NoticeEdit />},
        ]
      }
    ]
  }
];
```

FullLayout.tsx

```
<Header handleOpen={handleOpen} />
<div className='mainContainer'>
  <div className='lmbContainer na'>
    <Lnb open={naviOpen} />
  </div>
  <div className='pageContainer'>
    <Outlet />
  </div>
</div>
```

localhost:3000/notice/list

localhost:3000/notice/detail/1