Qcast coding convention

요구 스펙

> ⓘ info
>
> node 18.19.x 이상 LTS
> npm 대신 yarn 사용 (버전 1.19.xx 이상)

어플리케이션 scaffolding

```
src
├── app
│   ├── RecoilWrapper.js
│   ├── UIProvider.js
│   ├── changelog
│   │   └── page.jsx
│   ├── favicon.ico
│   ├── globals.css
│   ├── intro
│   │   └── page.jsx
│   ├── layout.js
│   ├── login
│   │   └── page.jsx
│   ├── page.js
│   ├── roof
│   │   └── page.jsx
│   └── roof2
│       └── page.jsx
├── components
│   ├── Headers.jsx
│   ├── Hero.jsx
│   ├── Intro.jsx
│   ├── Roof.jsx
│   ├── Roof2.jsx
│   ├── area
│   │   └── button
│   ├── fabric
│   │   ├── HelpLine.js
│   │   ├── QLine.js
│   │   ├── QLine2.js
│   │   ├── QPolygon.js
│   │   ├── QPolygon3.js
│   │   └── QRect.js
│   └── ui
│       └── RangeSlider.jsx
├── hooks
│   ├── useCanvas.js
│   └── useMode.js
├── lib
│   ├── canvas.js
│   └── prisma.js
├── middleware.js
├── pages
├── store
│   └── canvasAtom.js
└── util
    ├── canvas-util.js
    ├── qline-utils.js
    └── qpolygon-utils.js
```

컴포넌트 명시

기본적으로 server component를 지향하지만 다음 내용에 해당할 경우 client component를 명시한다.

- 써드파티 라이브러리를 사용할 경우
- 이벤트 핸들러를 작성하여 이벤트를 컨트롤 해야 할 경우
- useState 또는 useEffect 같은 내장 hook나 custom hook등을 사용해야 할 경우

다만 되도록이면 server component를 유지하되 client component 부분만 따로 컴포넌트를 import 하는 방향을 지향한다.

파일 확장자

마크업을 리턴하는 컴포넌트나 파일등은 확장자를 .jsx로 한다. 아마 대부분 `/src/app` 또는 `/src/components` 가 여기에 해당한다.
나머지 파일등은 그대로 .js 확장자를 사용한다.

파일명 & 함수명

`/src/app` 하위에 위치하며 마크업을 리턴하는 파일들은 대부분 페이지에 랜딩되는 파일이기 때문에 파일명이 `page`로 고정이 되어있다.
이 파일에 위치하는 default 함수명은 `파스칼 케이스`로 작성하고 `Page` 접미사를 사용한다.

```
export default function AboutPage() {

}
```

이 케이스를 제외한 일반 js 파일들의 파일명과 함수명등은 기존대로 `카멜케이스`를 사용한다.

커스텀 컴포넌트

페이지를 작성하다 보면 마크업 부분이 길어져서 파일이 커지는 경우가 많은데 재사용가능한 부분 또는 client component가 될 일부분의 경우 등은 커스텀 컴포넌트로 분리하여 작성한다. 이런 커스텀 컴포넌트는 `/src/components` 하위에 위치한다.

커스텀 훅스

리액트 내장 훅스를 제외한 특정 함수 등을 커스텀 훅스로 분리하여 작성하면 여러가지 장점을 제공받을 수 있다. 이런 커스텀 훅스는 `use` 접두사를 사용하고 `/src/hooks` 하위에 위치한다.
custom hook

유틸리티

유틸리티 파일들은 util 접미사를 사용하고 `/src/utils` 하위에 위치한다.

환경변수

개발환경의 환경변수는 .env.development 파일에 위치하고 키의 이름은 제한이 없지만 client component에서 불려질 변수는 `NEXT_PUBLIC_` 접미사를 사용해야 사용이 가능하다. 개발환경은 yarn dev 명령으로 실행한다.

운영환경의 환경변수는 .env.production 파일에 위치하고 키의 이름은 제한이 없지만 client component에서 불려질 변수는 `NEXT_PUBLIC_` 접미사를 사용해야 사용이 가능하다. 개발환경은 yarn build && yarn start 명령으로 실행한다.

.env.xxx 패턴의 파일이 .env를 덮어쓰기 때문에 .env 파일에는 공통 변수가 위치하고 .env.development 와 .env.production 파일에는 서로 오버라이드 될 변수를 사용한다.

development 환경에서 오버라이드 되는 순서

.env.development → .env.local → .env

production 환경에서 오버라이드 되는 순서

.env.production → .env.local → .env

상태관리

많은 상태관리 라이브러리가 있지만 본 프로젝트에서는 recoil을 사용한다.
파일은 store 접미사를 사용하고 `/src/store` 하위에 위치한다.

비동기통신

client component에서 비동기 통신을 위해 axios를 사용한다. 요청타입에 따라 함수를 래핑해 두었으니 맞는 함수를 임포트해서 사용한다.

```JS
import { get } from '@/lib/Axios'

const getUsers = async () {
        const users = await get('/api/users')
        console.log(users)
}
```

로그인

로그인 프로세스에 한해서 `prisma orm` 을 사용한다.
M_USER 테이블에 직접 접근하여 로그인 프로세스를 진행하고 서버세션을 생성해 인증을 관리한다.