물방개 코딩 컨벤션

파일 / 폴더

- 소문자
- kebab-case

```
user-profile/
  use-auth.ts
```

컴포넌트

- PascalCase

```
//components/button.tsx
export function Button() {...}
```

변수 / 함수

- carmelCase

```
const isLogged = true
const fetchData = () => {}
```

boolean

- is / has

```
isLoading
hasError
```

React / Next 규칙

type

- 공통으로 2군데 이상 사용되는 type은 type 폴더에 따로 작성 후 export
- 한 군데에서만 사용되는 type은 해당 파일 상단에 작성

컴포넌트 구조

- export default는 page, layout 단위만 사용

```
// app/page.tsx

export default function Main() {
    return ( ... )
}
```

- 컴포넌트는 export function 사용

```
//components/common/ImageContainer.tsx
export function ImageContainer() {
    return ( ... )
}
```

APP Router

- app/ 하위에는 라우팅 구조만 담당
- app/ 바깥에는 로직

```
src/
├ app/
   ├--profile/
         ├--page.tsx  //여기가 라우팅 페이지 진입점
├ components/      // UI 단위
├ features/        // 기능 단위
   ├--profile/
      ├--profile-view/
├ hooks/
├ lib/
├ types/
```

app 내부에서는 최종 조립만 담당합니다

다음과 같은 내용은 app에 두지 않습니다.

- 상태관리로직
- API호출
- 공용 hook

정석적인 app 사용 방법 예시입니다.

```
// app/profile/page.tsx
import { profileView } from '...'

export default function Page() {
   return <profileView />
}
```

```
// features/profile/profile-view.tsx

export function profileView() {
    const profile = useProfile() //여기에서 데이터를 받아오고
    return (
        ...코드를 작성합니다.
    )
}
```

라우트 별 layout을 써도 되는 경우

App Router 구조는 기본적으로 라우트마다 layout을 생성할 수 있지만

필수는 아니며, 필요한 경우에만 선택적으로 생성합니다.

1. 하위 모든 페이지에 공통 UI 필요

```
dashboard/
 ├ layout.tsx
 ├ page.tsx
 └ settings/page.tsx
```

2. 레이아웃 자체가 필요로 하는 데이터

> 여기서 children이 없어도 의미가 있는지?

```
export default async function Layout({ children }) {
  const menus = await getMenus()

  return (
    <>
      <Nav menus={menus} />
      {children}
    </>
```

```
  )
}
```

3. 인증/권한 가드

```
export default async function Layout({ children }) {
  const user = await getUser()

  if (!user) redirect('/login')

  return children
}
```

4. metadata / SEO 공통 처리

```
export const metadata = {
  title: 'Dashboard',
}
```

스타일

스타일 컴포넌트

1. 스타일 컴포넌트 파일 분리

```
🚫절대 금지🚫

const Wrapper = styled.div`
  /* 100줄 */
`
```

```
export function Page() {
  return <Wrapper />
}
```

```
user-card/
 ├ user-card.tsx
 └ user-card.styles.ts
```

```
// user-card.tsx
import * as S from './user-card.styles'

export function UserCard() {
  return <S.Wrapper />
}
```

1-1. 만약 분리하기에 스타일 컴포넌트 양이 적은 경우, **상단이 아닌 하단**에 작성 부탁드립니다.

2. styled는 순수 스타일만 담당하도록

가독성, 우선순위 등의 문제로 분리하는 게 좋을 것 같습니다.

```
const Box = styled.div`
  color: ${({ active }) => active ? 'red' : 'blue'};
`
```

지금까지는 이런 식으로 props를 전달해서 조건부 처리를 했는데

가독성 및 코드 복잡성이 많이 상승하는 것 같습니다.

JSX컴포넌트와 스타일 컴포넌트의 책임을 분리하는 게 좋을 것으로 판단됩니다.

아래는 올바른 예시입니다.

```
<Box className={isActive ? 'active' : ''} />

const Box = styled.div`
  &.active {
    color: red;
  }
`
```

3. 공통 스타일 분리

```
styles/
 ├ theme.ts
 ├ mixins.ts
 └ global.ts
```

공통으로 분리한 스타일의 경우 팀원에게 공유 부탁드립니다.

애매한 경우도 상의해서 넣었으면 좋겠습니다.

4. tailwind / styled 사용 우선순위

현재 tailwind도 설치 한 상황입니다.

- 반복 UI → tailwind
- 복잡한 인터랙션, 재사용되는 컴포넌트 스타일 → styled

```
display: flex
align-items: center
justify-content: center

혹은

position: absolute
top: 20px
```

```
혹은

width: 100vw
height: 100vh
```

예를 들어 다음과 같은 경우는 tailwind가 훨씬 간단합니다.

Import 순서

```
//1 . React 관련
import { useState } from 'react'
import Image from 'next/image'

//2. 외부 라이브러리
import clsx from 'clsx'
import styled from 'styled-components'

//3. 내부 공용 모듈
import { useAuth } from '@/hooks/use-auth'
import { getUser } from '@/lib/api'

//4. 비즈니스 로직
import { ProfileView } from '@/features/profile'
//5. 컴포넌트 UI
import { Button } from '@/components/button'
//6. style, assets
import * as S from './user-card.styles'
```

그리고, 만약 삭제되었거나 경로 변경된 파일이 있으면 나중에 몰아서 처리하지 말고

삭제or변경 즉시 체크해서 반영해주세요.

---

git commit 전 체크

```
npm run build
```

실행해서 type error, 경로 error 등 문제 사항 등을 꼭 체크해주세요.

---

git branch 전략

기존 피시마켓과 동일하게 dev 브랜치 아래 각자 이슈 생성 후 작업합니다.

- 기능단위 : feature : { 내용 }
- 에러 : error : { 내용 }
- 스타일 : style : { 내용 }

커밋 전략

커밋 전략은 조금 변경했습니다.
이슈 번호 적지 않고 컴포넌트 단위로 적는게 나중에 찾기 편할 것 같습니다.

```
//example

location : 검색 버튼 클릭 시 에러 예외처리 추가
```

현재 내가 작업중인 page 혹은 컴포넌트 단위 : 코딩 내용