6인 개발 팀원들에게 바로 공유하고, 프로젝트의 README.md나 팀 위키(Notion 등)에 바로 붙여넣으실 수 있도록 **[프런트엔드 개발 표준 가이드라인]** 형태로 정리해 드립니다.

📑 프런트엔드 개발 표준 및 설계 가이드

이 문서는 프로젝트의 유지보수성, 재사용성, 팀 협업 효율을 극대화하기 위한 순수 기능 중심**(Pure Feature-based)** 아키텍처와 **SPA** 설계 원칙을 정의합니다.

---

1. 프로젝트 구조 (Pure Feature-based)

우리는 모든 비즈니스 로직을 기술 계층(Component, Hook)이 아닌 기능**(Feature)** 단위로 응집시킵니다.

Plaintext

```
src/
├── app/              # 앱 진입점 (Providers, Router, Global Styles)
├── pages/            # 라우트 페이지 (Feature들을 조합하여 화면 구성)
├── features/         # [핵심] 독립적인 비즈니스 기능 단위
│   ├── [feature-name]/ # 예: auth, game-board, video-chat
│   │   ├── components/ # 해당 기능 전용 UI
│   │   ├── hooks/      # 해당 기능 전용 로직 (Composable)
│   │   ├── api/        # 해당 기능 전용 API 요청
│   │   ├── types/      # 해당 기능 전용 타입 정의
│   │   └── index.ts    # 외부 노출을 위한 Public API (Entry Point)
├── shared/           # [공용] 어디서든 사용 가능한 범용 요소
│   ├── components/     # 디자인 시스템 (Button, Input, Modal 등)
│   ├── hooks/          # 범용 유틸리티 훅 (useInterval, useDebounce 등)
│   └── utils/          # 순수 자바스크립트 함수
└── assets/           # 정적 리소스 (Images, Icons)
```

---

2. 핵심 개발 원칙 (The Golden Rules)

✅ **Rule 1:** 자기 완결성 **(Self-Contained)**

- 특정 기능을 수정할 때 해당 features/[name]/ 폴더 밖을 벗어나지 않는 것을 지향합니다.
- **로직(Hooks) + UI(Components) + 데이터(API)**는 항상 한 세트로 움직입니다.

✅ **Rule 2:** 엄격한 캡슐화 **(Public API)**

- 각 feature는 내부 구현을 숨기고 index.ts를 통해서만 소통합니다.
- **Bad:** import { Button } from '@/features/auth/components/Button'
- **Good:** import { LoginForm } from '@/features/auth' (index.ts에서 export된 것만 사용)

✅ **Rule 3:** 수평 참조 금지 **(No Horizontal Dependency)**

- features/auth가 features/game을 직접 참조할 수 없습니다.
- 공통 로직이 필요하다면 상위 레벨인 shared/로 격상시키거나, pages/에서 두 기능을 조합하세요.

---

3. SPA 코드 설계 원칙

성능과 사용자 경험(UX)을 위해 다음 4가지 원칙을 준수합니다.

원칙	설명	실천 방안
Logic 분리	UI 컴포넌트와 비즈니스 로직을 분리	복잡한 상태/이펙트는 반드시 **Custom Hooks**로 추출
SSOT 유지	데이터의 단일 출처 원칙 준수	**Server State**(React Query)와 **Client State**를 엄격히 분리
코드 분할	초기 로딩 속도 최적화	모든 Page 컴포넌트는 React.lazy를 통한 **Lazy Loading** 적용
선언적 코드	'어떻게'보다 '무엇을'에 집중	Suspense와 Error Boundary를 활용해 선언적으로 상태 처리

---

4. Feature 개발 예시 (Template)

팀원들이 참고할 수 있는 features/ 내부 작성 표준입니다.

TypeScript

```
// features/auth/hooks/useAuth.ts (Composable 로직)
export const useAuth = () => {
  const [user, setUser] = useState(null);
  const login = async () => { /* 로그인 로직 */ };
  return { user, login };
};

// features/auth/index.ts (외부 노출 정의)
export * from './hooks/useAuth';
export { LoginForm } from './components/LoginForm';
export type { User } from './types';
```

---

5. 팀 협업 체크리스트

1. 새 기능을 만드나요? features/ 아래에 새로운 폴더를 생성하세요.
2. 공통 버튼인가요? shared/components/에 있는지 확인하고 만드세요.
3. 파일이 너무 긴가요? UI는 컴포넌트로, 로직은 훅으로 즉시 분리하세요.
4. 다른 기능의 코드가 필요한가요? 직접 참조하지 말고 pages/에서 조합하거나 shared/를 활용하세요.

팀장으로서의 한마디:

> "우리의 목표는 응집도는 높이고 결합도는 낮추는 것입니다. 각자의 feature 폴더 안에서는 자유롭게 개발하되, 폴더 밖으로 나가는 연결 고리는 최소화하여 서로의 코드에 영향이 가지 않도록 합시다!"

이 내용을 기반으로 프로젝트를 시작하시면 팀원들이 각자 맡은 도메인에 집중할 수 있어 개발 속도가 훨씬 빨라질 것입니다.

다음 단계로, 이 구조에서 사용할 전역 상태 관리 도구(Zustand, Recoil 등)나 서버 데이터 라이브러리(TanStack Query) 설정 방법을 정리해 드릴까요? 구체적인 스택이 정해졌다면

그에 맞춘 베스트 프랙티스 예시 코드를 짜 드릴 수 있습니다.