Next.js + FSD(Feature-Sliced Design)

next.js 가 꼭 필요한가?

개념 설명

1. 클라이언트 사이드 렌더링(CSR) vs 서버 사이드 렌더링(SSR) vs 사전 렌더링 (PreRendering)

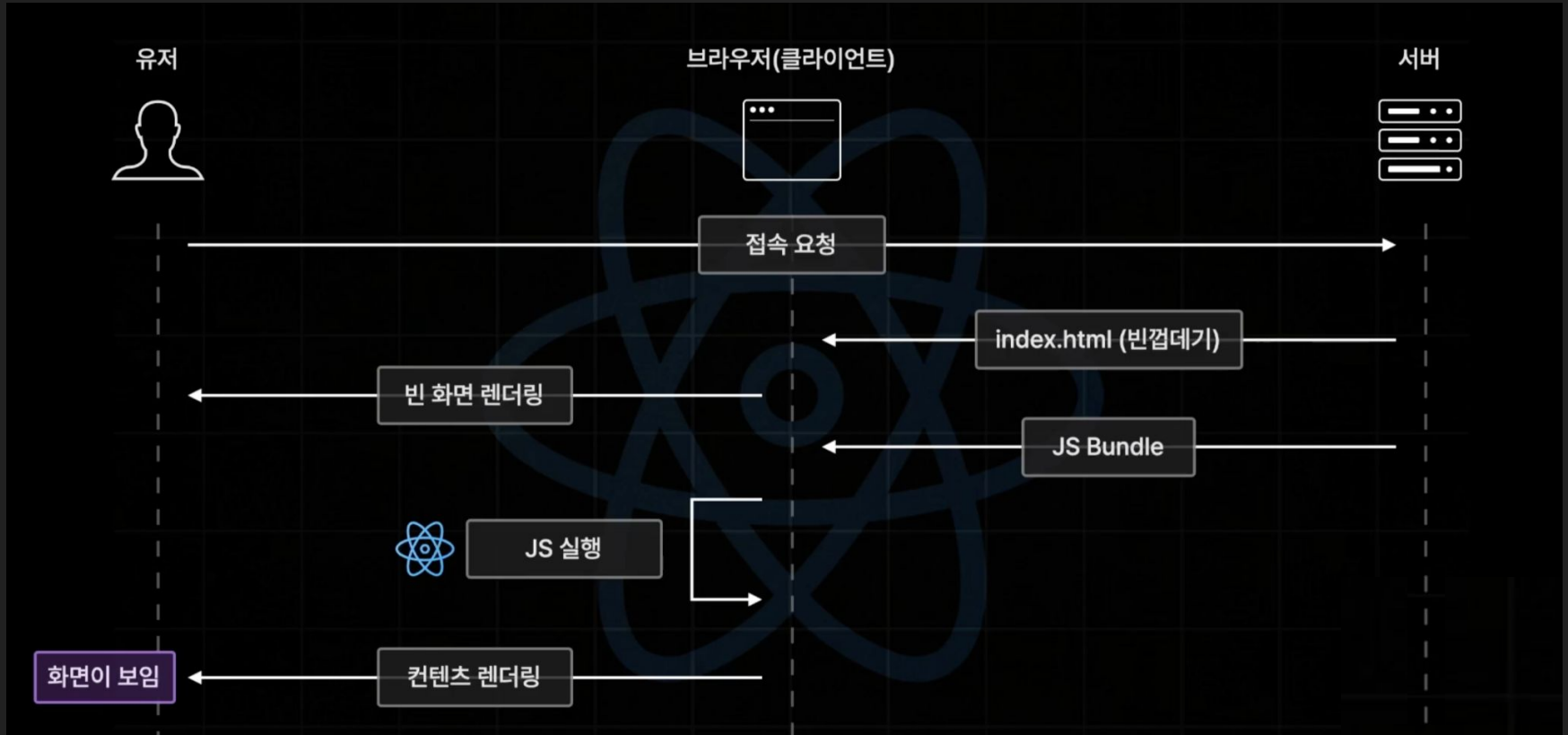
2. Next.js 의 서버 사이드 렌더링

3. FSD(Feature-Sliced Design)의 구조 설명

클라이언트 사이드 렌더링 (CSR)

- React 앱의 기본적인 렌더링 방식
- 클라이언트(브라우저)가 직접 렌더링을 담당하는 방식

- 초기 접속 속도가 느리다
- FCP, 요청 시작 부터 컨텐츠가 렌더링 될때까



서버 사이드 렌더링 (SSR)

- Next.js의 기본적인 렌더링 방식

- 서버에서 완전히 렌더링된 HTML 페이지를 생성하여 클라이언트에게 전송하는 방식

NextJS 란?

사전 렌더링(Pre-Rendering)

빌드 시점에 미리 HTML 파일을 생성하여 서버에 배포하는 방식

- SSR (Server-Side Rendering) : 이 방식은 사용자의 요청에 따라 각 페이지를 실시간으로 생성합니다. 요청이 있을 때마다 서버에서 HTML을 생성하여 클라이언트로 보냅니다. 다이나믹한 데이터를 다루는 페이지에 적합합니다.
- SSG (Static Site Generation) : SSG는 빌드 시점에 모든 페이지를 HTML로 미리 생성합니다. 각 페이지가 사용자의 요청 이전에 이미 생성되어있으므로, 요청 시 서버에서 바로 제공할 수 있습니다. 변경되지 않는 데이터를 표시하는 페이지에 적합합니다.

Next.js

FSD

(Feature-Sliced Design)

- pages layer 라우팅 파일

- NextJS에서 app layer의 미지원 또는 충돌

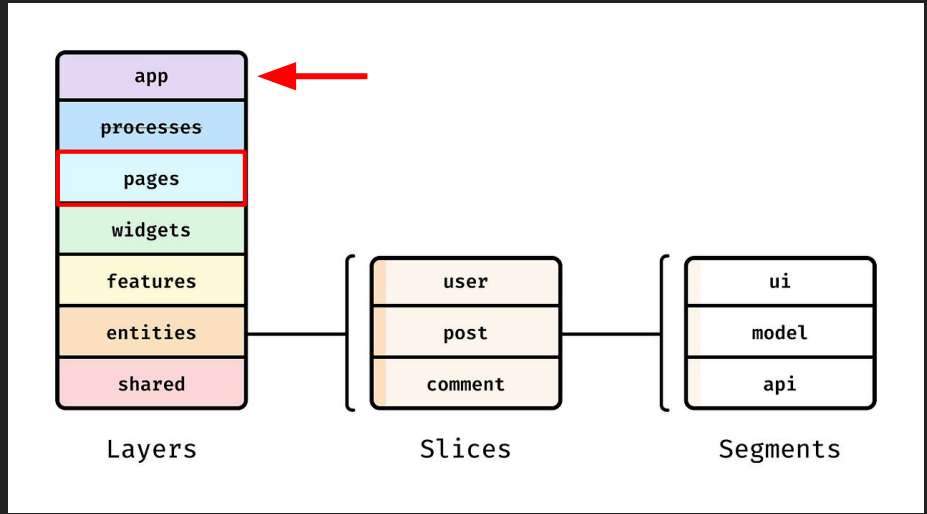
NextJS와 함께 사용하기 - 문제 1) pages layer 라우팅 파일

Next.js

- Next.js의 pages/ 는 “라우팅 엔진” 폴더 (파일 = URL)

- Next.js는 파일 시스템 기반 라우팅

FSD



- FSD의 pages 레이어는 “페이지 모듈” 레이어 (모듈 = 화면)

- pages 레이어에 페이지들을 flat한 slice로 둔다

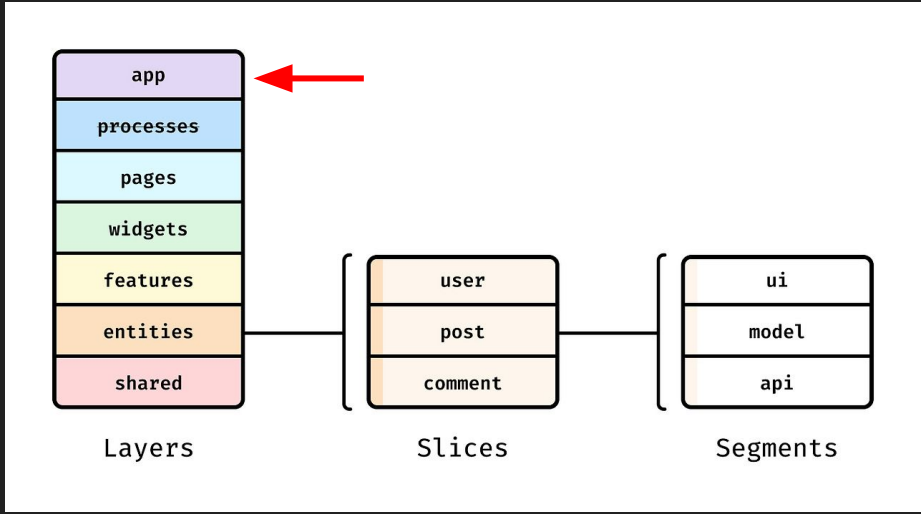
Next.js

Next.js (Framework)
페이지 라우팅 기능을 구현해야 함

Next.js 이 제공하는 라우터

- Page Router
- App Router (13+)

FSD



* FSD의 app layer을 넣을 표준 폴더가 아니라 _app.tsx가 엔트리

* Next의 app/layout.tsx은 라우트, FSD의 app은 초기화

- pages layer 충돌

- 제 1안) Next.js pages 폴더를 Project Root로 이동

```
├── pages                # NextJS 라우팅 폴더 (FSD pages를 재-export)
│   └── index.tsx
│   └── about.tsx
├── src
│   ├── app
│   ├── entities
│   ├── features
│   ├── pages            # FSD pages layer
│   ├── shared
│   └── widgets
```

- pages layer 충돌
- 제 1안) Next.js pages 폴더를 Project Root로 이동
- 제 2안) FSD pages layer 이름을 변경

```
├── app
├── entities
├── features
├── pages                # NextJS 라우팅 폴더
├── views                # 변경된 FSD pages layer
├── shared
├── widgets
```

Page Router : FSD app layer에 대응하는 폴더 X

Page Router : FSD app layer에 대응하는 폴더 X

App Router

```
├── app                 # NextJS의 App Router용 폴더
├── pages               # NextJS의 Pages Router용 폴더 (선택적)
│   ├── README.md       # 폴더의 용도 설명
├── src
│   ├── app             # FSD의 app layer
│   ├── entities
│   ├── features
│   ├── pages           # FSD의 pages layer
│   ├── shared
│   ├── widgets
```

Page Router : FSD app layer에 대응하는 폴더 X

App Router

Middleware

- Next.js middleware 파일은 반드시 프로젝트 root 폴더(app 또는 pages 폴더와 동일 수준)에 둬야함

- src 아래에 두면 Next.js가 인식 X

```
# FSD 아키텍처 구조

doctorstock-web/
├── src/
│   ├── app/                          # 라우팅/메타/레이아웃은 여기서만 결정
│   │   ├── (public)/                 # route group (URL에 안 잡힘)
│   │   │   └── login/
│   │   │       └── page.tsx
│   │   ├── (protected)/
│   │   │   └── home/
│   │   │       └── page.tsx          # 서버 컴포넌트(기본)
│   │   ├── layout.tsx
│   │   ├── providers.tsx             # 전역 Provider (client)
│   │   └── page.tsx
│   │
│   └── layers/                       # FSD 레이어들 (app 밖에 둬서 라우팅과 분리)
│       ├── app/                      # ✅ FSD app layer (전역 Provider/Styles/엔트리 로직)
│       │   ├── providers/
│       │   ├── styles/
│       │   │   └── globals.css
│       │   └── api-routes/        # (선택) API 로직(핸들러 구현부) :contentReference[oaicite:3]{index=3}
│       │
```

```
├── pages/                    # "페이지 모듈" (Next 라우트 파일 아님!)
│   ├── home/
│   │   ├── ui/
│   │   │   └── HomePage.tsx
│   │   ├── lib/              # 유틸리티 함수, slice내에서 사용되는 보조 기능
│   │   ├── consts/           # 설정값, 상수 값들을 담당
│   │   ├── model/            # 상태 관리 & 비즈니스 로직, 즉 상태와 상호작용, action 및 selectors가 해당됨
│   │   └── api/              # API 호출 로직
│   │
│   └── login/
│       ├── ui/
│       │   └── LoginPage.tsx
│       ├── lib/              # slice내에서 사용되는 보조 기능, util 함수의 성격
│       ├── consts/           # 필요한 상수 값들을 담당
│       ├── model/            # 비즈니스 로직, 즉 상태와 상호작용, action 및 selectors가 해당됨
│       └── api/              # 필요한 서버 요청 담당
│
├── widgets/
│   └── header/
│       ├── ui/
│       │   └── Header.tsx    # 헤더 컴포넌트
│       └── model/
│           └── useNavigation.ts  # 네비게이션 로직
```

NextJS와 함께 사용하기 - 제안하는 구조

```
├── features/                    # 로그인 기능이나 비밀번호 일치 여부 검증, 일정 규칙을 따르는지 검증 로직 등이 해당
│   ├── auth/
│   │   └── login/
│   │       ├── ui/
│   │       │   └── LoginForm.tsx
│   │       ├── model/
│   │       └── api/
│   │
│   └── onboarding/
│       ├── ui/
│       └── model/
│
├── entities/                    # FSD Entities 레이어(User 엔터티를 정의하여 사용자 데이터를 관리)
│   └── user/
│       ├── ui/
│       ├── model/
│       ├── api/
│       └── lib/
```

```
└── shared/                  # Button과 같은 기본 컴포넌트를 제공, 프로젝트 전반에서 다양한 버튼 스타일을 일관되게 사용
    ├── ui/                  # 기존의 components
    │   ├── Button/
    │   │   └── Button.tsx   # 공통 버튼
    │   ├── Input/
    │   │   └── Input.tsx    # 공통 인풋
    │   └── Modal/
    │       └── Modal.tsx    # 공통 모달
    ├── api/
    │   ├── client.ts        # 클라에서만 쓰는 요청
    │   ├── server/          # 서버에서만 쓰는 fetch
    │   └── common/          # 둘 다 가능한 공용
    ├── lib/                 # 공통 유틸리티 함수 및 헬퍼 함수
    │   ├── formatDate.ts    # 날짜 포맷팅 함수
    │   ├── mathHelpers.ts   # 수학 계산 관련 헬퍼 함수
    │   └── api.ts           # 공통 API 호출 함수
    ├── config/
    │   ├── apiUrls.ts       # API URL 목록
    │   └── roles.ts         # 사용자 권한 관련 상수 값
    ├── styles/
    │   └── theme.css        # 전역 테마 설정 (컬러, 폰트 등)
    ├── types/
```

NextJS와 함께 사용하기 - 제안하는 구조

```
│           ├── types/
│           ├── hooks/                # 재사용 가능한 React Hooks
│           │   ├── useWindowSize.ts # 윈도우 크기 감지 Hook
│           │   └── useDebounce.ts   # 디바운스 기능을 제공하는 Hook
│           └── auth/                 # 인증 관련 공통 로직
│               ├── utils.ts         # 유틸 함수
│               └── validation.ts    # 검증 로직
│
├── public/                           # Assets, Fonts
└── package.json
```

참고 : 공식문서 (https://stackblitz.com/edit/stackblitz-starters-aiez55?file=src%2Fpages%2Fhome%2Findex.ts)