ChatGPT

프로젝트 컨텍스트 분석 기반 멀티모달 문서 자동화 플랫폼 – UI/UX 구현 명세서

1. 정보 구조(IA) & 내비게이션

Problem: 이 플랫폼은 관리자, PM, 실무자, 외부감사(조회전용) 등 여러 역할별로 접근하는 **엔터프라이즈 웹 앱**입니다. 각 역할마다 볼 수 있는 메뉴와 기능이 달라야 하며, 방대한 문서와 프로젝트를 효율적으로 찾고 탐색할 수 있어야 합니다. 잘못된 역할에 잘못된 메뉴가 노출되면 **보안 위험**과 **혼란**이 초래될 수 있습니다. 또한 사용자는 현재 자신의 **위치**(정보 구조 상 어디에 있는지)를 늘 알아야 하며, 대용량 데이터를 다루는 만큼 **검색 기능**을 통한 빠른 접근이 필수입니다.

Principle: 역할에 따른 **최소 권한 원칙**을 적용하여 **필요한 메뉴만 노출**하고, 불필요한 인지 부하를 줄입니다. UI에서는 권한 없는 기능을 아예 숨기거나 비활성화하여 혼동을 예방하지만, **실제 권한 검사는 백엔드에서 강제**해야 합니다[1]. **항상 탐색 가능**한 내비게이션 구조를 갖춰 사용자가 길을 잃지 않도록 합니다. Nielsen Norman Group의 메뉴 가이드에 따르면 **데스크톱에서는 주요 내비게이션 메뉴를 항상 가시적으로 표시**하여 사용자가 제품의 범위와 구조를 파악하도록 해야 합니다[2]. 현재 위치는 메뉴 하이라이트와 **브레드크럼**으로 표시하여 "지금 내가 어딨지?"에 답을 줍니다[3]. 또한 **글로벌 검색**은 `/` 단축키로 바로 포커스되어 프로젝트, 문서, 청크, 태그 등을 통합 검색할 수 있게 해 **직접 탐색 시간을 최소화**합니다.

Pattern: 상단 고정 AppBar에는 제품 로고, 화면 제목, 글로벌 검색창(돋보기 아이콘+placeholder "Search…"), 우측에 사용자 아바타/설정/알림 아이콘을 배치합니다. **좌측 사이드바**는 역할별로 다른 메뉴 세트를 갖춘 **영구 사이드 내비게이션**으로 구현합니다 (데스크톱에서는 항상 펼쳐진 형태). 예를 들어 관리자에게는 모든 메뉴가, 실무자에겐 본인 작업 관련 메뉴만 보입니다. 메뉴 항목에 아이콘과 레이블을 부여하고, 현재 활성 메뉴는 강조 표시합니다. 화면 상단에는 해당 위치의 **브레드크럼**을 표시하여 상위 프로젝트나 섹션으로 쉽게 이동 가능하게 합니다. 사이드바 하단에는 **역할 전환/로그아웃** 등의 부가 메뉴를 둡니다. **글로벌 검색**은 입력 시 결과 미리보기를 드롭다운으로 보여주고, 엔터 시 검색 결과 페이지로 이동합니다. 단축키는 헤드업 디스플레이(HUD) 도움말 또는 설정 페이지에 안내하고, 예: `G P` -> 프로젝트 대시보드, `U` -> 업로드 화면, `V` -> 벡터 검색, `L` -> 로그 에디터, `T` -> 템플릿 매핑, `A` -> 승인 화면 등으로 정의합니다.

Implementation: 각 **역할(Role)**에 대한 **메뉴 가시성/행동 권한 매트릭스**를 수립합니다. 예를 들어 외부감사는 **조회만 가능**하도록 모든 쓰기 버튼을 숨기고 메뉴도 읽기 기능만 남깁니다. 반면 관리자는 모든 프로젝트/설정 메뉴에 접근 가능합니다. 이 매트릭스를 토대로 프론트엔드 라우팅 가드와 Supabase RLS 규칙을 함께 설계합니다. UI 레벨에서는 Oso 등의 권고처럼 권한 없는 버튼을 표시하지 않거나 비활성화하여 "사용 불가" 상태를 명시하되, 결정적으로 **API 단계에서 차단**하여 일관성을 유지합니다[1]. 내비게이션은 **React Router**와 같은 SPA 라우터를 사용하여 사이드바 메뉴와 URL 경로를 연결합니다. 브레드크럼은 URL 경로 혹은 라우트 메타정보를 이용해 생성하고, **aria-label**을 적용해 스크린리더가 "현재 위치: 프로젝트 > 상세" 등을 읽도록 합니다. 글로벌 검색은 Debounce를 걸어 **2초 내 검색 완료**(KPI)하도록 백엔드(pgvector) 질의 최적화 및 인덱싱을 설정하고, 검색 결과에는 문서/프로젝트 아이콘과 제목, 하이라이트된 일치 키워드 등을 표시합니다. 키보드 내비게이션을 위해 `/` 눌러 포커스 이동, 검색 결과 목록은 화살표로 탐색 + 엔터로 이동 가능하도록 구현합니다.

2. 컬러 · 타이포그래피 · 레이아웃 디자인 토큰

Problem: Corporate Blue 중심 팔레트를 사용하는 일관된 브랜드 UI를 구현해야 합니다. 동시에 WCAG 2.2 AA 명도 대비 기준을 충족하여 **접근성**을 보장해야 합니다. 개발과 디자인 팀이 동일한 언어로 소통하려면 색상, 글꼴, 여백 등 **디자인 토큰**을 정의하고 코드에 반영해야 합니다. 현재 제시된 Primary 색 (`#0B63F6`) 및 Success/Warning/

Danger 등 보조 색상, 그리고 4px 기준의 spacing 스케일 등이 있는데, 이를 **Tailwind CSS** 구성에 정확히 맵핑할 필요가 있습니다. 다양한 컴포넌트에서 padding, radius 등이 일관되게 쓰이도록 토큰화하고, **라이트 테마**로 시작하되 향후 다크 테마나 테마 변경도 고려해야 합니다.

Principle: 디자인 토큰을 통해 **단일 소스의 진실(single source of truth)**로 스타일을 관리합니다. 디자인 토큰은 색상, 폰트, 간격 등의 핵심 디자인 값을 이름으로 정의하여 **디자이너와 개발자가 공통 언어**로 사용할 수 있게 합니다 [4]. Tailwind CSS의 설정 파일(`tailwind.config.js`)이 곧 디자인 시스템의 토큰 저장소로 기능하며, 여기서 정한 값만 UI에 쓰이도록 합니다. 토큰은 **안정적이고 최소한으로** 정의합니다 – 너무 세분하면 혼란을 초래하므로, 색상도 핵심 단계(예: 50, 100, …, 900)만 사용하고 간격도 4px 기준 **의미있는 단계**만 만듭니다 [5] [6]. 또한 **명명 규칙**을 직관적으로 합니다. 예를 들어 `--color-primary-600`처럼 토큰명이 **의도**를 담도록 하고 단순 숫자 대신 의미 중심 이름(`primary`, `surface`, `text`)으로 그룹화합니다 [7]. 접근성을 위해 **대비 대비**도 중요합니다. WCAG 기준 일반 텍스트는 **4.5:1**, 큰 텍스트는 **3:1** 이상의 명도비가 권장되므로 [8], Primary 600 위에 흰 텍스트를 쓸 경우 대비비를 검증해야 합니다. 또한 **색상만으로 상태를 표시하지 않고**, 아이콘이나 패턴을 함께 사용하도록 (예: 에러 메시지는 빨간색 굵은 글자 + 아이콘) 디자인 토큰에 **status 색상과 함께 사용될 아이콘** 정보도 포함시킵니다.

Pattern: Primary Palette는 제시된 `#0B63F6`(600) 기준으로 50부터 900까지 **10단계**를 설계합니다. 예를 들어 50은 매우 옅은 청색 (`#E8F2FF` 정도), 900은 거의 검정에 가까운 짙은 청색으로 설정합니다. Primary 600 자체는 UI 핵심 색으로 버튼 배경 등에 쓰이며, **Hover** 시 약 10% 어둡게(`#0953CC`), **Pressed** 시 20% 어둡게(`#0742A3`) 사용하는 식으로 계층화합니다. On-Primary (전경 글씨)로는 **흰색**을 사용하되 투명도나 톤 조정을 통해 대비를 확보합니다. Secondary, Success (`#16A34A` 초록), Warning (`#F59E0B` 주황), Danger (`#DC2626` 빨강), Info (`#0EA5E9` 하늘) 색상도 각자 500~600대의 기준 색과 밝은/어두운 변형을 정의합니다. **Neutral 계열**은 Gray 50, 100 … 950까지 Material3의 Neutral 팔레트를 참고하여 설정합니다 (예: Gray 50=`#F9FAFB`, 900=`#111827`). 배경은 기본 흰색 (`#FFFFFF`), 표면(카드 등)은 약간 다른 흰색 (`#F9FAFB` 등), 보더는 Gray 200, 분리선은 Gray 300 등으로 지정합니다. **폰트**는 Pretendard를 기본 sans-serif로 쓰며, 코드나 키보드 단축키 표시에는 Monospace (JetBrains Mono 등) 글꼴을 사용합니다. 폰트 사이즈 토큰은 `text-sm`=14px(20px line-height), `text-base`=16px(24px), `text-lg`=20px(28px), `text-xl`=24px(32px), `text-2xl`=28px(36px) 등으로 설정합니다. **Spacing** 토큰은 4px 단위를 기반으로 `spacing-1`=4px, 2=8px, 3=12px, 4=16px, 5=20px, 6=24px, 8=32px, 10=40px, 12=48px, 16=64px 등으로 정의합니다 [6]. `border-radius` (라운드) 토큰은 `radius-sm`=4px, `md`=8px, `lg`=16px, `xl`=24px로 적절히 제한합니다 (버튼은 8px, 카드 16px, 모달 24px 등). **Shadow**(그림자) 토큰도 `shadow-sm`, `md`, `lg` 3단계로 정의해 엘리베이션을 표현합니다 (예: `shadow-sm`: 0 1px 2px rgba(0,0,0,0.05), `shadow-md`: 0 2px 4px rgba(0,0,0,0.1) 등). 모션 관련 토큰은 `motion-duration-default`=150ms, `motion-duration-enter`=200ms, `motion-duration-exit`=150ms와 같이 정의하고 `motion-easing`은 `cubic-bezier(0.2, 0.8, 0.2, 1)`로 통일합니다. 또한 사용자 **선호도("모션 감소")** 설정 시 animation이나 parallax를 줄이는 조건(`prefers-reduced-motion`)을 CSS에 반영합니다.

Implementation: 디자인 토큰은 우선 JSON 또는 TS 객체로 정리하고, Tailwind 설정에서 이를 **커스터마이징**합니다. 예를 들어 `tailwind.config.js`의 `theme.extend`에 아래처럼 추가합니다:

```
// tailwind.config.js (일부 발췌)
module.exports = {
 theme: {
  colors: {
   primary: {
    50: '#E8F2FF',
    100: '#BFDDFF',
    ...,
    600: '#0B63F6',  // Primary base
    700: '#0953CC',
```

```
      800: '#0742A3',
      900: '#062f73'
    },
    success: '#16A34A',
    warning: '#F59E0B',
    danger:  '#DC2626',
    info:    '#0EA5E9',
    gray: {
      50: '#F9FAFB',
      100: '#F3F4F6',
      ...,
      900: '#111827'
    }
  },
  spacing: {
    1: '4px', 2: '8px', 3: '12px', 4: '16px', 5: '20px',
    6: '24px', 8: '32px', 10: '40px', 12: '48px', 16: '64px'
  },
  borderRadius: {
    sm: '4px', md: '8px', lg: '16px', xl: '24px', full: '9999px'
  },
  boxShadow: {
    sm: '0 1px 2px 0 rgba(0,0,0,0.05)',
    md: '0 2px 4px 0 rgba(0,0,0,0.1)',
    lg: '0 8px 16px 0 rgba(0,0,0,0.1)'
  },
  fontFamily: {
    sans: ['Pretendard', 'ui-sans-serif', 'system-ui', 'sans-serif'],
    mono: ['ui-monospace', 'SFMono-Regular', 'Menlo', 'monospace']
  },
  fontSize: {
    sm: ['14px', '20px'],
    base: ['16px', '24px'],
    lg: ['20px', '28px'],
    xl: ['24px', '32px'],
    '2xl': ['28px', '36px']
  }
  // ... 기타 설정
 }
}
```

위와 같이 Tailwind에 **커스텀 토큰**을 정의하면, 개발자는 `bg-primary-600`, `text-gray-800`, `rounded-lg`, `p-4` 등 **유틸리티 클래스**로 디자인 토큰을 손쉽게 활용할 수 있습니다 [9]. 또한 Tailwind v3+의 CSS 변수 기반 테마 기능(@theme)을 활용하면 토큰을 CSS 변수로 노출할 수도 있습니다 [10] [11]. 예를 들어 `--color-primary-600` 변수를 정의해두면 자바스크립트나 CSS에서 동적으로 해당 색상을 사용할 수도 있습니다. 최종 산출물로 **design-tokens.json** 파일을 제공하여 (예: 색상 팔레트, 타이포, 스페이싱 등) 디자이너와 개발자가 참고하도록 하고, 동 내용이 Tailwind 설정에 반영되었음을 명시합니다. **명도 대비 검사**는 Adobe Contrast Checker 등을 통해 토큰 정의 단계에서 수행하고 [8], 예를 들어 Primary 600 배경에 흰 텍스트의 대비비가 4.5:1 이상인지 확인합니다. 만약 부족하면 Primary 700을 텍스트 배경으로 쓰거나 폰트 사이즈를 키우는 식으로 조정합니다. 이렇게 정의된 토큰은 추후 다크모드 테마 확장이나 디자인 변경시 해당 JSON과 Tailwind 설정만 업데이트하면 전체 앱에 일관되게 반영되므로 **유지보수 효율성**이 높습니다.

3. 컴포넌트 규격서 (사이즈, 아이콘, 상태)

Problem: 이 플랫폼에는 카드, 버튼, 입력 필드, 테이블, 탭, 모달 등 다양한 UI 컴포넌트가 존재합니다. 일관되지 않은 컴포넌트 구현은 **사용자 경험의 혼란**과 유지보수 어려움을 초래합니다. 따라서 각 컴포넌트의 크기, 여백, 아이콘 위치, 그리고 **다양한 상호작용 상태(기본, 호버, 포커스, 비활성, 로딩, 에러 등)**를 명확히 정의해야 합니다. 또한 10,000개 이상의 행을 다룰 수 있는 테이블 등 **성능 요구사항**도 있습니다. 컴포넌트 디자인에서 **접근성**을 고려하여 포커스 표시, 에러 시 시각+텍스트 표시 등이 필요합니다 (예: 에러를 단순 색상만으로 표시하지 않고 아이콘/텍스트로 중복 표시).

Principle: 일관성(consistency)과 **명확한 피드백**이 핵심 원칙입니다. 동일한 유형의 컴포넌트는 애플리케이션 전반에서 동일한 모양과 동작을 보여야 합니다. 예를 들어 모든 1차 액션 버튼은 모서리 반경, 폰트 크기, hover 색상 변화 등이 같아야 사용자도 예측 가능합니다. 컴포넌트는 **크기 체계**를 갖춰 Small/Medium/Large 등으로 구분하고, 모바일에서는 터치 목표를 고려해 충분한 크기(최소 44px 높이 또는 24px24px 이상)를 유지해야 합니다 [12] . 상태 변화에 대한 피드백 – 호버 시 강조, 포커스 시 뚜렷한 테두리, 로딩 시 스피너 – 를 제공하여 시스템 상태를 가시화합니다. 특히 에러 상태는 사용자 작업 흐름을 방해하므로, NN/g 가이드에 따라 눈에 잘 띄고 (highly visible), 구체적인 원인과 해결책을 텍스트로 제공해야 합니다 [13] [14] . 에러 메시지는 굵은 빨간 텍스트와 함께 아이콘, 하이라이트 등을 조합해 표시하고, 색상 하나에 의존하지 않도록 해야 합니다 [15] . 즉, 시각적 표시(예: 경고 빨간색 테두리)와 함께 텍스트와 아이콘*을 사용하여 누구나 오류를 인지할 수 있게 합니다 [15] .

Pattern:
- **버튼 (Button):** Primary, Secondary, Tertiary, Danger, Ghost 다섯 가지 **버전**을 정의합니다. Primary 버튼은 파란색 배경에 흰 텍스트 (브랜드 컬러)로 주된 액션에 사용하고, Secondary는 경계선만 있는 아웃라인 스타일 등으로 구분합니다. 버튼 **크기**는 `sm` (높이 28px, 폰트 13px, 좌우 패딩 8px), `md` (36px, 폰트 14px, 패딩 12px), `lg` (44px, 폰트 16px, 패딩 16px) 세 가지로, 모두 최소 터치 영역 44px을 만족하거나 내부 padding으로 이를 충족시킵니다. 아이콘이 있는 버튼은 아이콘 크기 16px으로 하고, 텍스트와 4px 간격을 둡니다. 버튼의 **상태**: 기본(default)은 불투명 100%, hover 시 약간 명도 다운(예: Primary #0B63F6 -> #0953CC), **focus**시 2px 두께의 포커스 링(파란색 50% 투명도 또는 대체 색상)으로 나타냅니다. WCAG 2.2 **Focus Appearance** 기준에 따라 이 포커스 링은 컴포넌트에 2px 테두리를 두른 크기이고 인접 색과 3:1 대비를 갖춰야 합니다 [16] . **비활성(disabled)** 상태는 텍스트/아이콘은 Gray 400, 배경은 Gray 100으로 명도 대비를 낮추고 커서는 기본 화살표로 합니다. **로딩** 상태는 버튼 안에 스피너를 표시하고 텍스트는 보여주되 상호작용을 막습니다 (또는 텍스트를 스피너로 대체). Danger 버튼은 Primary와 같지만 빨간색 계열을 사용하고, 파괴적 행동엔 추가 confirm 모달과 함께 사용됩니다.

    - **입력 필드 (Forms):** TextField(단일 행), TextArea(여러 행), Number, Date, Select, Combobox(자동완성), FileDrop(파일 드래그&업로드 영역), Tag Input 등으로 나눕니다. **레이블**은 상단에 14px 폰트로 표시하고, 선택 사항(optional)은 레이블 옆에 (선택) 등을 표시합니다. 필수 필드는 `*` 별표를 붙이거나 색상을 다르게 해 시각적으로 구분합니다. **플레이스홀더**는 회색(Gray 500)으로 짧게 예시를 보여주며, 중요한 경우 툴팁이나 도움말 아이콘으로 추가 설명을 제공합니다. **포커스** 시 필드 테두리를 Primary 600색 2px로 강조하고, shadow 효과를 줄여도 좋습니다. 입력 길이 제한이 있을 경우 (예: 200자) 하단에 현재/최대 글자수를 표시합니다. **유효성 검증**은 블러 시점이나 제출 시점에 수행하며, 에러 발생 시 해당 필드 아래에 빨간색 작은 글씨로 오류 메시지를 표시합니다. 이때 `lucide:alert-circle` 아이콘(16px)을 메시지 왼쪽에 표시해 **텍스트 외 시각 표시**를 추가합니다. 오류가 있는 필드의 테두리도 빨간색으로 하이라이트하고 아이콘을 필드 오른쪽 끝에 표시할 수도 있습니다 [15] . 에러 메시지는 사용자가 문제를 이해하고 해결할 수 있도록 **사람 친화적 언어**로 작성합니다 (예: "필수 정보입니다" 또는 "유효한 이메일을 입력해주세요") [14] . 성공적인 검증 후에는 해당 아이콘을 체크 표시로 표시하거나 메시지를 제거합니다.

    - **카드 (Card):** 대시보드 등에서 정보 패널로 쓰이는 카드 컴포넌트는 그림자와 패딩이 적용된 사각형입니다. **밀도**는 보통형(기본 패딩 16px)과 밀집형(8px) 두 가지를 두어 상황에 맞게 사용합니다. 카드의 **헤더** 영역에는 아이콘 + 제목 + 액션 메뉴(예: “:” 더보기 메뉴)를 배치할 수 있습니다. **본문**은 가변 영역으로 표나 텍스트, 차트를 담습니다. **푸터**가 있다면 보조 정보나 버튼(예: "상세보기")을 배치합니다. 카드의 radius는 16px (라운드 lg), 그림자는 `shadow-md`로 살짝 띄워진 효과를 줍니다. 카드 자체는 클릭 가능할 경우 hover시에 `shadow-lg`로 상승 효과와 약간의 변색(또는 테두리 강조)을 주어 인터랙티브함을 표시합니다.

- **테이블 (Table):** 데이터 리스트는 표 형태로 표시합니다. **머리글(Header)** 행은 배경을 Gray 100으로 하고 Bold 폰트로 구분합니다. 테이블은 세로 스크롤이 길어질 수 있어 **헤더를 상단에 고정**합니다. 또한 10,000행 이상에서도 부드럽게 작동하도록 **가상 스크롤(Virtualization)** 기법을 사용합니다 – 즉, 화면에 보이는 행만 DOM에 렌더링하고 나머지는 버퍼로 관리하여 성능을 확보합니다 [17]. 이를 통해 방대한 행 처리 시에도 브라우저 렉을 줄입니다. 칼럼 너비는 내용에 맞게 기본 자동 조절하되 사용자가 경계선을 드래그하여 **열 너비 조정**을 할 수 있게 합니다. 특정 칼럼(예: ID나 이름 등)은 좌측 고정(**freeze**) 옵션을 제공하여 스크롤 시에도 항상 보이게 합니다. **정렬/필터**: 헤더 셀에 정렬 아이콘을 두어 오름/내림차순 정렬 기능을 넣고, 필요한 경우 칼럼 필터(텍스트 검색 또는 드롭다운)를 지원합니다. **선택 기능**: 맨 왼쪽에 체크박스 열을 두어 다중 선택을 지원하며, 상단에 선택된 항목 개수와 일괄동작 버튼(삭제 등)을 표시합니다. **빈 상태**: 테이블에 보여줄 행이 없을 경우, 본문에 "데이터가 없습니다" 등의 메시지를 표 중앙에 표기하고 가이드 링크(예: "새 프로젝트를 생성하세요")를 함께 제공합니다 [18]. 단순 빈 공간만 두지 않아야 사용자가 오류와 빈 결과를 구분할 수 있습니다 [19] [18]. **오류 상태**: 데이터를 불러오지 못한 경우 오류 메시지와 재시도 버튼을 표 자리에 표시합니다. 페이지네이션(Pagination)은 하단에 제공하여 한 페이지당 50행 기본, 사용자가 페이지 이동이나 페이지당 개수 변경을 할 수 있게 합니다. 키보드 내비게이션(위/아래 방향키) 및 스크린리더 호환성을 위해 `<table><thead><tbody>`의 올바른 마크업과 `<th scope="col">`, `<th scope="row">` 사용을 준수합니다.

- **탭 (Tab) 및 스텝퍼(Stepper):** 화면 내 국소적인 탭 전환이 필요하면 Material3의 탭 패턴을 사용합니다. 탭은 상단에 탭 리스트로 배치하고 선택된 탭은 강조(밑줄 또는 배경색)합니다. **키보드 포커스**로 좌우 이동 가능하도록 `<button role="tab">` 구조를 사용합니다. **스텝퍼**는 업로드→파싱→검수→매핑→미리보기→결재→완료까지 프로세스 진행을 시각화하는 요소입니다. 진행 단계에 따라 **현재 단계**는 강조색 테두리와 번호/아이콘 표시, 완료된 단계는 체크 아이콘, 향후 단계는 회색 번호로 표시합니다. 사용자가 진행상태를 한눈에 파악하고, 완료 이전 단계는 클릭해 돌아갈 수 있지만 (권한이 있다면) 완료된 이후 단계는 접근 제한하는 등의 **가드**를 둘 수 있습니다. 모바일에서는 탭이나 스텝이 많을 경우 가로 스크롤 가능하게 하거나, 단계별 요약을 아코디언 형태로 전환합니다.

- **모달/다이얼로그 (Modal & Drawer):** 중요 확정 동작이나 설정 변경은 **모달 대화상자**로 띄웁니다. 모달은 화면 중앙에 나타나며 배경을 반투명하게 dimming 처리합니다. **라지 모달**은 `max-width:600px` 정도, **풀스크린 모달**은 모바일 또는 필요한 경우 전체화면으로도 사용합니다. 확인/취소 버튼을 모달 하단 우측에 배치하고, 파괴적 액션은 빨간 **위험(Danger)** 버튼으로 표시하며 이중 확인을 위한 경고 메시지도 함께 넣습니다. 예를 들어 "정말 삭제하시겠습니까? 이 작업은 되돌릴 수 없습니다."와 같은 문구와 취소/삭제 버튼을 제공합니다. **드로어**는 오른쪽 측면에서 슬라이드 인하는 패널로, 템플릿 매핑 스튜디오 등 **화면 대부분을 차지해야 하는 업무**에 사용합니다. 드로어는 닫기 "×" 버튼 및 Esc 키로 닫을 수 있고, 내부 스크롤이 가능하며, 뒤에 배경 dimming은 하지 않습니다(필요시 반투명 처리). 모달과 드로어 모두 **포커스 트랩**을 구현하여 열린 상태에서 Tab키 포커스가 그 안에서만 돌도록 하고, ARIA `aria-modal="true"`, `role="dialog"` 등을 지정합니다.

- **토스트(Toast) 알림:** 짧은 피드백 메시지는 화면 우측 하단에 **토스트**로 표시합니다. 예를 들어 "업로드 성공", "저장 완료" 등의 성공 메시지는 **3초** 후 자동 사라지게 하고, 경고는 5초, 오류는 8초로 더 오래 노출합니다. 토스트에는 아이콘과 함께 요약 메시지를 담고, 사용자 조치가 필요한 경우 클릭하면 상세 로그 패널을 열거나 (예: "자세히 보기") 할 수 있습니다. 예컨대 오류 토스트에 "자세히" 링크를 제공해 해당 작업의 로그 화면으로 이동시킵니다. 여러 토스트가 쌓일 경우 최신이 가장 위에 오도록 하고, 5개 이상이면 스크롤 가능하거나 자동으로 가장 오래된 것부터 소멸시킵니다. 또한 **모션 감소** 설정 사용자의 경우 토스트 등장 애니메이션(슬라이드 인 등)을 최소화합니다.

- **토글 및 기타 폼 요소:** Toggle 스위치는 이진 상태 설정에 사용하며, 체크박스 대용으로 **ON/OFF 시각 구분이 명확**하게 디자인합니다. (예: ON일 때 내부 토글 원이 우측 끝, 배경 파랑; OFF일 때 좌측, 배경 회색). 크기는 40px x 20px 정도로 하고, 포커스 시 뚜렷한 파랑 테두리 표시를 합니다. 일반 **체크박스**는 16px 정사각형에 모서리 2px 라운드, 체크 아이콘은 SVG로 넣고, **라디오 버튼**은 16px 원형으로 구현합니다. 이들도 클릭 가능한 주변 레이블까지 합쳐 터치 영역 24px 이상 확보하고, 포커스 링 적용 및 **키보드 Space로 토글 가능**하게 처리

합니다 [12]. **배지(Badge)**는 상태나 숫자를 표시하는 작은 레이블로, 배경색은 회색 또는 상태 색, 글자는 12px로 하고 모서리 완전 라운드(full)로 칩 형태로 만듭니다. **아바타(Avatar)**는 사용자 이니셜 또는 프로필 이미지를 원형으로 표시하며 작은 경우 24px, 기본 32px, 큰 경우 40px 등으로 정의합니다. 이미지가 있을 때와 없을 때 대비 색상(Gray 300 배경에 이니셜 등)을 지정합니다. **페이징(Pagination)** 컴포넌트는 '<', '>' 및 숫자 버튼들로 구성하며, 현재 페이지는 강조 표시하고 비활성 페이지 버튼은 aria-disabled 속성을 둡니다. 키보드로도 Page Up/Down으로 넘길 수 있게 고려합니다.

Implementation: Storybook 등을 활용해 모든 컴포넌트의 변형(variants)과 상태를 문서화합니다. **컴포넌트 prop 시그니처**와 CSS 클래스를 정의하여 개발팀에 전달합니다. 예를 들어 `<Button variant="primary" size="md" icon="plus" disabled />` 형태로 사용할 수 있게 하고 각 variant마다 Tailwind 유틸리티를 적용하거나 headless UI 방식으로 스타일을 입힙니다. 포커스 스타일은 Tailwind의 `focus:ring-2 focus:ring-primary-500 focus:ring-offset-2` 같은 클래스로 통일합니다 (outline 제거는 하지 않음: `outline:none` 금지 [20]). 에러 메시지의 경우 `<Field errorMessage="..." />` prop으로 에러 상태를 주입하면 내부적으로 아이콘+텍스트를 렌더링하도록 컴포넌트를 구성합니다. 10K 행 이상의 테이블은 React 가상화 라이브러리(e.g., `react-virtual` 또는 Ag-Grid)를 적용합니다. 실제 구현에서 [TanStack Virtualizer] 등의 오픈소스를 활용하여 보이는 영역만 DOM에 유지함으로써 **브라우저 메모리**와 **리플로우**를 최적화합니다 [17]. 필드 컴포넌트들은 label과 input을 `<label>`로 감싸 클릭 영역을 넓히고, aria-invalid, aria-describedby 등을 통해 접근성 속성을 넣습니다. 각 컴포넌트의 hover, focus, disabled, error 등의 CSS 클래스는 디자인 토큰 색상을 참조하여 구현하며, 예컨대 `.btn-primary:hover { background-color: #0953CC }` 식으로 Tailwind 플러그인이나 `@apply` 지시자를 활용할 수 있습니다. **키보드/스크린리더 테스트**를 병행하여 Tab 순서, role, aria-label 등이 빠짐없도록 하고, **WCAG 폼 접근성 가이드** (오류시 포커스 이동, 오류 메시지 연결 등)를 따른다 [21] [22]. 이처럼 정의된 컴포넌트들은 재사용하여 개발 생산성을 높이고, Figma에도 동일한 스펙으로 라이브러리로 제공하여 디자인-개발 싱크를 맞춥니다.

4. 대시보드 & 업로드/파싱/임베딩

Problem: 대시보드는 전체 프로젝트와 시스템 상태를 한눈에 보여주는 관문입니다. 현재 대기 중인 작업, 진행률, 평균 처리 시간, 최근 실패한 작업 등을 표시해야 합니다. 사용자가 즉시 상태를 파악하고 필요한 조치를 취할 수 있어야 합니다. **업로드** 모듈에서는 최대 200MB 파일(동시 5개까지)을 업로드하고 파싱하는 긴 과정을 다룹니다. PDF, PPTX, 이미지, 텍스트 등 다양한 포맷을 지원해야 하며, 특히 200페이지 PDF 파싱은 수분(최대 5분) 걸릴 수 있습니다. 따라서 사용자에게 **진행 상황을 지속적으로 보여주고** 기다리는 동안 불안을 줄이는 UI가 필요합니다. 또한 파일에 개인정보(PII)가 포함될 수 있어 업로드 시 이에 대한 경고와 마스킹 안내를 제공해야 합니다. 동시에 많은 파일을 올릴 때 **서버 과부하**나 실패 가능성도 있으므로, 오류 처리와 재시도 기능도 중요합니다.

Principle: 한눈에 보기 (Glanceability)와 **명확한 진행 피드백**이 중요합니다. 대시보드는 사용자에게 현재 시스템의 **중요 지표(KPI)**를 빠르게 전달해야 하므로 시각적으로 강조된 숫자, 차트 등을 활용합니다. 업로드/파싱 과정에서는 Nielsen의 **"시스템 상태 가시화"** 원칙을 따라 사용자가 지금 어떤 단계에 있고 얼마나 남았는지 알 수 있게 해야 합니다. 진행 표시 디자인은 **스켈레톤 vs. 진행률 표시** 기준을 명확히 합니다. 불확실한 로딩에는 스켈레톤을 사용해 **골격 화면**을 보여주어 기다림을 덜 지루하게 하고 [23], **예측 가능한 작업(파일 업로드, 변환 등)**에는 남은 시간이나 %를 알려주는 **Progress Bar**를 사용합니다 [24]. 특히 업로드/변환처럼 **소요 시간을 어느 정도 알 수 있는 경우** skeleton보다 **명시적 진행률**이 낫습니다 [24]. 사용자는 장시간 작업일수록 중간에 **취소**하거나 다른 일을 볼 수 있어야 하므로, 작업 중 취소/일시정지 후 재개 옵션을 제공합니다. 또한 PII 관련 **보안 원칙**으로, 민감 정보가 포함될 가능성을 경고하고 필요 시 **자동 마스킹** 옵션을 제안합니다.

Pattern:
- **대시보드 화면:** 상단에 **프로젝트 요약 카드**들을 배치합니다. 예: "총 프로젝트 수", "이번 주 업로드 문서 수", "승인 대기 문서" 등의 통계를 **숫자 + 라벨** 카드로 나타냅니다. KPI 목표와 연동하여 "평균 생성 시간: 2m 30s (목표 3m 이내)"처럼 목표 대비치를 뱃지로 표시할 수 있습니다. 메인 섹션에는 **파이프라인 상태 차트**를 넣습니다: 예를 들어 진행중/대기/완료/오류 건수를 원형 차트나 막대그래프로 시각화합니다. 최근 활동 피드(업로드 완료, 승인됨 등)를 시간순

리스트로 보여줘 **감사 로그**와는 별개로 사용자에게 최근 움직임을 피드백합니다. 또한 최근 오류가 발생한 작업을 카드로 강조(빨간색 아이콘과 함께 "프로젝트 X - PPTX 생성 실패")하고 클릭 시 상세 로그나 재시도로 이동하게 합니다.

- **업로드 큐 UI:** 사용자가 파일을 선택하거나 드래그하면 업로드 대기열에 아이템으로 나타납니다. 각 아이템(파일)에 대해 썸네일(파일 유형 아이콘), 파일명, 크기, 상태, 진행률을 보여줍니다. **동시 업로드 5개** 제한이 있으므로, 사용자가 한 번에 6개 이상 드롭하면 "5개까지 동시 업로드, 초과 파일은 대기열에 추가됩니다" 등의 메시지를 토스트로 알립니다. 파일당 진행률 바가 있고, 전체 진행률(총 n개 중 x개 완료)도 별도로 표시합니다. 개별 **취소/일시정지** 버튼을 제공하여 사용자가 특정 업로드를 취소할 수 있게 합니다. PII 경고: 업로드 UI 상단에 배너로 " 개인식별정보(PII)가 포함된 문서는 업로드 전에 마스킹이 필요할 수 있습니다. [자세히 보기]"를 표시하여 사용자 인지를 높입니다. 이 배너는 관리자 권한일 때만 보이거나 모든 사용자에게 보이도록 설정 가능합니다.

- **진행 및 상태 표시:** 각 파일 업로드 아이템 옆에 **실시간 상태**를 텍스트로 표시합니다. 예: "업로드 중 (45%)...", "파싱 대기", "파싱 중 (페이지 30/200)", "벡터 임베딩 중 (75%)", "완료 ", "오류 ". 진행 중인 단계는 **스피너** 아이콘을 함께 돌리고, 완료 시 체크 마크, 실패 시 느낌표 아이콘으로 표시합니다. **긴 작업(파싱, 템플릿 주입 등)**에는 예상 남은 시간(ETA)을 표시합니다. 예를 들어 "파싱 중 (3분 예상 남음)" 같이 보여주어 사용자가 기다릴 시간을 가늠하도록 합니다. 만약 ETA 계산이 어렵다면 진행 퍼센트와 함께 "남은 시간 계산 중..."처럼 안내합니다. 5분 이상 걸리는 작업의 경우 사용자에게 **다른 작업을 진행할 수 있음을 안내**하거나, 백그라운드에서 진행되고 완료 시 알림을 주겠다는 메시지를 표시합니다 ("문서 파싱에는 수 분이 소요될 수 있습니다. 다른 작업을 하셔도 좋습니다. 완료되면 알려드리겠습니다."). 이러한 경우 대시보드의 작업 현황판이나 알림 센터에 해당 작업이 등록되어 진행 상태를 언제든 확인할 수 있게 합니다.

- **오류 처리 & 재시도:** 업로드한 파일 중 파싱 실패나 형식 오류가 발생하면, 해당 아이템에 빨간색 오류 아이콘과 함께 "실패: PDF 파싱 오류" 등의 원인을 표시합니다. 또한 **"재시도"** 버튼을 함께 제공하여 클릭 시 해당 파일을 다시 처리하도록 요청합니다. 연속 실패 시에는 (2~3회) 사용자가 다른 조치를 취할 수 있도록 "지원 팀에 문의" 링크나 "로그 다운로드" 버튼을 넣습니다. 전반적인 큐 처리 중 서버나 네트워크 에러로 중단되면, 전체 업로드 영역 상단에 배경 빨간 에러 배너를 띄워 "서버와 연결이 끊어졌습니다. 인터넷 연결을 확인 후 재시도하세요." 등을 알립니다. 이때 사용자가 **데이터 손실 없이 재개**할 수 있도록, 업로드 큐 상태를 보존하고 재시도 버튼을 클릭하면 이어서 진행하도록 설계합니다.

- **스켈레톤 로딩:** 대시보드 진입 시 API 호출 지연으로 데이터가 바로 없으면, 숫자 카드나 차트 영역에 **스켈레톤 UI**를 표시합니다 (회색으로 반짝이는 자리표시바). 예를 들어 KPI 숫자 대신 회색 블록, 차트 대신 연한 사각형 등을 보여줍니다 [23]. 1-2초 내에 데이터가 도착하면 skeleton이 실제 내용으로 부드럽게 대체됩니다. 반면 업로드 진행처럼 **시작이 명확히 트리거된 작업**은 skeleton보다는 구체적 진행 표시를 바로 보여주는 것이 좋습니다.

Implementation: 대시보드 지표들은 백엔드에서 집계 API를 통해 가져옵니다. 프론트엔드는 해당 API 응답을 받아 **Recharts** 등의 라이브러리로 그래프를 그리고, 실패율이 높다면 red, 성공률 높으면 green 등 색상으로 인디케이터를 표시합니다. 업로드 컴포넌트는 `<input type="file" multiple>`와 drag & drop 이벤트를 활용하여 구현하고, 선택된 File 객체를 배열로 관리하면서 UI 목록을 렌더링합니다. 각 파일 업로드는 **XHR** 또는 Fetch로 진행하며 `XMLHttpRequest.upload.onprogress` 등을 통해 % 진행을 갱신해 state에 반영합니다. 파싱/임베딩 등의 서버 처리 진행은 Supabase 함수 호출이나 Job 큐를 통해 처리되므로, 프론트는 주기적으로 진행 상태를 폴링하거나 WebSocket 채널(Subscribing)로 실시간 업데이트를 받습니다. (예: Supabase Realtime이나 다른 채널 사용). 진행률 바 컴포넌트는 % 값을 prop으로 받아 길이를 계산하며, ARIA `role="progressbar"`와 `aria-valuenow` 등을 넣어 스크린리더에도 진행률을 읽을 수 있게 합니다. **Skeleton**은 Tailwind의 animate-pulse 클래스를 활용하거나 커스텀 SVG로 만들어 placeholder 자리에 넣습니다. Skeleton은 콘텐츠 레이아웃을 미리 보여주되 내용은 비워두는 형태로, 사용자가 느끼기에 로딩 시간을 20~30% 단축된 것처럼 느끼게 해줍니다 [25]. 그러나 파일 업로드/변환처럼 **정확한 진행값을 아는 경우**에는 skeleton 대신 **Determinate Progress bar**를 사용하기로 한 원칙을 지킵니다 [24]. 업로드 완료 또는 실패 등의 이벤트 발생 시, 해당 사실을 토스트로도 알려주고 (특히 사용자가 다른 화면에 있을 경우 대비), 대시보드의 최근 활동에 항목 추가, 실패 시 오류 로그 DB에 기록 등 후속 처리를 합니다. PII 경고 배너는

환경설정에서 on/off 가능하게 하고, [자세히 보기] 클릭 시 개인정보 보호 지침 페이지로 연결하거나 마스킹 기능 설명 모달을 팝업합니다. 마지막으로, 이러한 긴 프로세스를 **로드 테스트**하여 5개 동시 200MB 파일 업로드 시 프론트엔드 UI 반응과 진행 표시가 원활한지 검증하고, 필요하다면 한 번에 일정 수 이상은 대기큐로 돌리는 등 **부하 제어 UI**도 고려합니다.

5. 벡터 검색 & 근거 뷰

Problem: 사용자는 파싱된 방대한 비정형 데이터에서 **원하는 정보를 빠르게 검색**하고, AI가 제시하는 결과의 **근거를 확인**하길 원합니다. 단순 키워드 검색이 아닌 벡터 유사도 기반 검색을 제공하므로, 검색 UI에서 **유사도 임계값 조절**이나 **필터링**이 필요합니다. 또한 검색 결과가 문서의 특정 페이지나 이미지 영역이라면, 그 **문맥과 하이라이트**를 함께 보여줘야 사용자가 신뢰할 수 있습니다. 예컨대 "프로젝트 X의 예산이 얼마인가?"를 검색하면, 관련 문서의 해당 부분을 표시하고, AI가 답변에 이용할 근거로 삼을 수 있어야 합니다. 따라서 검색 UI는 단순 결과 리스트가 아니라 **하이라이트된 스니펫, 점수/신뢰도 배지, 그리고 원본 미리보기**를 갖춘 형태여야 합니다.

Principle: 맥락 속 검색 결과(Contextual Results)와 **결과에 대한 신뢰도(Confidence) 제공**이 핵심입니다. 사용자에게 검색 결과가 왜 나왔고 얼마나 관련있는지 설명해야 하며, 필요하면 **출처를 직접 검증**할 수 있게 해야 합니다. 이를 위해 **하이라이트**된 키워드나 문장이 포함된 스니펫을 보여줘야 합니다 [26] . 또한 결과마다 유사도 점수를 기반으로 **신뢰도 배지**(High/Medium/Low 또는 % 점수)를 부여해, AI 답변 생성시 참고 가능성을 판단할 수 있게 합니다. 예를 들어 유사도 0.92 이상은 "높음", 0.75~0.9는 "중간" 배지 (색상으로도 구분: 녹색, 노랑 등)로 표시합니다 [27] . **UX 패턴**으로는 “Hover to reveal source”, “Split view”, “Highlight and scroll” 등이 고려됩니다 [28] . 즉, 화면이 협소할 땐 마우스오버로 원문을 툴팁처럼 보여주거나, 넓을 땐 왼쪽에 Q&A/로그, 오른쪽에 문서뷰어를 띄워 **양쪽을 동시에 볼 수 있는 split view**를 제공합니다 [28] . 또한 검색 결과의 특정 문장을 클릭하면 아래 근거 패널의 PDF/PPT 페이지가 **해당 문장으로 스크롤**되어 하이라이트를 깜빡이게 합니다 [28] . 이러한 상호작용으로 사용자에게 “왜 이 결과가 선택되었는지” 투명하게 보여주는 것이 신뢰성을 높입니다.

Pattern:
- **검색 쿼리 빌더:** 검색창 옆에 **고급 옵션 토글**이나 “필터” 아이콘을 제공하여, 클릭 시 필터 패널을 보여줍니다. 여기서 **프로젝트 필터**(드롭다운 또는 태그로 다중 선택), **파일 형식 필터**(pdf/pptx 등), **날짜 범위** 설정(업로드일 기준), 그리고 **유사도 임계값 슬라이더**(예: 0.0 ~ 1.0 조정)를 제공합니다. 기본 임계값은 0.8로 두고 사용자 조정에 따라 결과 범위를 늘리거나 줄일 수 있게 합니다. 검색어 입력은 키워드 또는 자연어 질문 모두 지원하므로 placeholder에 “키워드 또는 질문 입력...”이라고 안내합니다. 검색 실행은 Enter로 가능하고, 바로 RAG질의를 보내 백엔드의 pgvector로부터 top-k 결과를 가져옵니다.

- **결과 리스트 UI:** 검색 결과는 각 **결과 카드** 형태로 표시합니다. 각 카드에는 **문서명** (또는 출처명을 식별할 수 있게 프로젝트명/파일명), 해당 문서에서 추출된 **스니펫 내용**을 보여줍니다. 스니펫은 사용된 임베딩 청크 단위로 하며, **검색어와 일치/유사한 키워드들을 하이라이트** 처리합니다 [26] . 예를 들어 “예산”을 찾았다면 문장 중 **예산** 글자를 노란 마커로 강조합니다. 한 결과에 여러 청크가 연속하면 적절히 `...`으로 줄이거나 다중 스니펫으로 나눕니다. **유사도 점수**는 카드 상단 혹은 제목 옆에 배지로 표시합니다. (예: “신뢰도 높음” 배지 혹은 점수 0.92). **아이콘 표시:** 결과가 텍스트 기반(pdf/txt)인지 이미지인지에 따라 아이콘을 다르게 표시해 사용자가 자료 유형을 알 수 있게 합니다. 이미지의 경우 썸네일이나 “[이미지 스니펫]”이라 쓰고, 클릭 시 이미지 뷰어로 보여주도록 합니다.

- **근거 미리보기 패널:** 화면 우측(혹은 하단)에 **근거(Evidence) 패널**을 둡니다. 사용자가 검색 결과를 클릭하면, 이 패널에 해당 결과의 **원본 문서 미리보기**를 표시합니다. PDF/PPTX인 경우 PDF.js 또는 PPT 뷰어를 내장하여 해당 페이지를 렌더링합니다. 이때 **해당 검색어가 하이라이트**된 채로 표시되고, 스크롤 위치를 자동으로 맞춥니다. 이미지인 경우 해당 이미지와 OCR 결과 텍스트를 함께 보여줄 수 있습니다. 만약 bounding box 정보가 있다면 이미지 내 강조 테두리로 표시합니다 (예: “예산” 글자가 있는 도형에 빨간 박스). 패널 상단에는 문서명, 페이지 번호 등을 표시하고 “문서에서 보기” 혹은 새 탭 열기 버튼을 제공하여 전체 문서를 열람할 수도 있게 합니다. 또한 검색 결과가 여러 개 선택되면 탭으로 패널 내 전환하거나, 한 번에 여러 근거를 비교 표시할 수도 있습니다.

- **신뢰도/필터 UI:** 검색 결과 상단에 **결과 정렬/필터 바**를 둡니다. 기본은 유사도 순 정렬이며, 사용자가 프로젝트별 그룹핑이나 최신순(메타데이터 기반)이 필요하면 옵션을 추가합니다. **유사도 임계값** 슬라이더는 예를 들어 0.8 이상 결과 5개를 가져왔을 때 더 보고 싶다면 낮추도록 UI에 안내합니다 ("더 많은 결과를 보려면 임계값을 낮추세요"). 또 신뢰도 배지에 마우스오버하면 "벡터 유사도 0.82 (임계값 0.8)"처럼 정확한 수치를 툴팁으로 보여줍니다.

- **"로그 에디터로 보내기":** 각 검색 결과 카드에는 **액션 버튼**으로 "+ 로그에 추가" 또는 "편집기에 보내기" 같은 옵션을 둡니다. 이를 클릭하면 현재 열린 멀티모달 로그 에디터(섹션 6) 쪽에 해당 근거를 인용하거나, 또는 AI 답변 초안에 이 내용을 포함시키도록 트리거합니다. 예를 들어 AI가 작성한 문단에 근거로 이 문서 부분이 연결되거나, 혹은 "이 내용 인용" 기능으로 해당 텍스트를 주석과 함께 삽입합니다. 이를 통해 검색-작성 경험이 분리되지 않고 이어지도록 합니다.

Implementation: 백엔드에서 pgvector similarity search API를 구축하고, 프론트엔드에서 검색 질의 입력시 REST 호출 또는 RPC 함수를 실행합니다. 검색 결과 데이터에는 문서 식별자, chunk 텍스트, 임베딩 점수, 페이지/위치 등이 포함되어 옵니다. 이를 받아 위 패턴대로 컴포넌트를 렌더합니다. **텍스트 하이라이트**는 JavaScript 정규식이나 highlight.js 같은 라이브러리를 써서 구현합니다. React에서는 `<mark>` 태그로 감싼 하이라이트 처리를 해주고, 색상은 디자인 토큰의 강조색(노랑 등)으로 합니다. PDF/PPT 미리보기는 PDF.js 라이브러리를 `<canvas>`로 렌더하거나, pdfobject로 embed하는 방식도 고려됩니다. PPTX의 경우 HTML5로 렌더링하는 라이브러리가 적은데, 서버측에서 이미지를 생성해 제공하거나, Python-pptx로 텍스트 추출하여 간이 뷰어로 제공할 수 있습니다. 이미지의 OCR 텍스트는 사전에 저장된 것이 있다면 같이 표시합니다. **Hover to reveal** 패턴은, 결과 리스트의 문구 위에 마우스를 올리면 툴팁으로 그 문맥의 더 긴 내용을 보여주는 기능입니다 [28]. 구현은 `<div title="...">` 기본 툴팁으로는 어렵고, 별도 tooltip 컴포넌트에 원문 텍스트 전체를 넣어 띄우는 식으로 합니다. Split view 모드는 화면 크기에 따라 반응형으로 전환합니다. 예를 들어 넓은 화면에서는 결과 리스트와 근거 패널을 나란히(`display:flex`) 보여주고, 좁은 모바일 화면에선 결과 항목을 누르면 근거 패널이 **모달**이나 새로운 화면으로 뜨게 합니다. **유사도 점수**는 백분율로 환산 가능하지만, 정확한 기준 이해를 돕기 위해 0.00~1.00 사이 수치를 0~100으로 표시하거나 Low/Med/High로 범주화합니다 (임계값 0.8을 High 경계로). 색상은 High=녹색, Medium=주황, Low=회색으로 설정합니다 [27]. 검색 속도(KPI 2초 이내)를 맞추기 위해 결과 건수를 5~10개로 제한하고, 필요한 경우 "더 보기" 버튼으로 추가 로드합니다. 프런트 상태관리는 React Query 등을 통해 검색어별 캐시를 두어 뒤로가기 시 재사용 가능하게 합니다. 마지막으로, **검색과 RAG 연계**: "로그 에디터로 보내기" 클릭 시 현재 선택 근거를 Redux 상태나 URL param으로 로그 에디터 페이지에 전달하거나, 에디터 컴포넌트에 직접 prop으로 전달합니다. 이때 해당 근거가 강조되어 나타나거나, AI가 이 근거를 사용해 문장 생성하게 API 호출 트리거하는 등 추가 구현을 합니다. 이러한 일련의 UI 흐름으로 검색→검증→활용이 하나의 UX 플로우로 연결되도록 만듭니다.

6. 멀티모달 로그 에디터

Problem: 멀티모달 로그 에디터는 AI가 생성한 기술 로그(문단들)를 사용자가 검토·수정하는 핵심 인터페이스입니다. 여기서는 텍스트뿐 아니라 이미지 캡션 등도 함께 다뤄지며, AI 생성 문장과 사용자가 편집한 최종 문장이 **혼재**합니다. 주요 과제는 (1) AI가 제안한 내용과 사용자가 수정한 내용을 비교하여 **변경 이력(diff)**을 보여주는 것, (2) 각 문장의 **출처(근거)**를 명시하여 사용자가 사실 여부를 검증할 수 있게 하는 것, (3) 문장별 **승인/거부** 액션을 통해 최종 산출에 들어갈 내용만 확정하는 워크플로우입니다. 또한 팀 협업을 위해 **코멘트 스레드** 기능으로 문장이나 섹션 단위 논의가 필요하고, **톤/용어 일괄 수정 규칙** 적용 등 편집 보조 기능도 요구됩니다. 모바일 환경에서는 현장 사진을 찍어 업로드하고 그에 대한 자동 캡션을 AI가 생성, PM이 승인하는 등의 시나리오가 고려됩니다.

Principle: 문서 편집기+검토의 혼합 UI 원칙이 적용됩니다. 사용자에게 AI 초안을 맹신하지 않고 검토하도록 유도해야 하므로, AI가 생성한 부분을 **명시적으로 표시**하여 신뢰도를 판단케 합니다. 이는 Google Docs의 제안 모드나 MS Word의 변경 추적처럼, **추가/삭제된 텍스트를 색상 차등**으로 보여주는 방식이 효과적입니다. 또한 각 문장에는 그 문장이 나온 **근거 링크/아이콘**을 붙여 사용자에게 "이 문장이 근거에 충실한가?"를 스스로 체크하도록 권장합니다 [29] [28]. UI는 **좌우 패널**로 정보와 편집 공간을 분리해, 왼쪽에는 편집 중인 로그 텍스트, 오른쪽에는 선택된 문장의 근거(이미지 또는 문서)를 동기화해 보여줍니다 [28]. **하단에는 코멘트 패널**을 둬 문단별 논의가 가능케 합니다. 이런 3분할 구조를 통해 편집-근거 검토-협업이 한 화면에서 이뤄집니다. 모든 UI 요소는 **멀티모달** (텍스트+이미지) 데이터를 다루는 데 맞

줘야 하므로, 예를 들어 이미지 캡션도 일반 문장처럼 편집 가능하고, 클릭하면 우측 패널에 해당 이미지 강조 표시 등 **텍스트-이미지 연계**가 이루어지게 합니다.

Pattern:
- **편집 영역(UI 좌측):** 각 로그 **문단** 또는 **항목**이 블록 단위로 나열됩니다. AI가 처음 생성한 문장은 일단 제안 상태로 표시됩니다. 이를 나타내기 위해 해당 텍스트에 연한 하이라이트(예: 파랑 테두리)를 두거나 AI 아이콘을 문장 앞에 배치할 수 있습니다. 사용자가 그 문장을 수정하면, 수정된 부분은 **diff 표시**로 이전 AI 문안과 비교됩니다: 삭제된 텍스트는 빨간 취소선, 추가된 텍스트는 녹색 밑줄 등으로 표현하거나, 간단히 색상차로 표현할 수 있습니다. (꼭 실시간 diff가 아니더라도, 편집 완료 후 "AI vs Edited" 토글로 비교 보기 기능 제공). 각 문단 오른쪽에는 액션 버튼들이 있습니다: **체크 표시**(승인), **X 표시**(제외) 버튼이 있어 해당 문장을 최종 산출물에 포함할지 결정합니다. 승인하면 그 문단은 녹색 라벨이 붙거나 상태 아이콘(체크)을 표시하고, 제외하면 회색 처리되며 취소선 또는 아이콘으로 표시됩니다. 물론 언제든 취소/재승인 가능하도록 토글 형태로 동작합니다. 상단에는 **일괄 규칙 적용** 드롭다운을 두어 "회사 용어집 적용"이나 "문어체/간결체 일괄 수정" 같은 기능을 실행할 수 있게 합니다. 이는 예를 들어 미리 정의된 금칙어나 문체규칙에 따라 AI 텍스트를 한 번에 교정하는 것으로, 실행 시 각 문장에 변경사항을 표시하고 강조합니다.

  - **근거 패널(UI 우측):** 편집 중 특정 문단을 클릭하면, 오른쪽 패널에 해당 문단과 연관된 **근거들**이 나타납니다. 예컨대 AI가 문장 "X의 예산은 Y억이다"를 생성했다면, 그 문장에 링크된 출처 PDF의 해당 페이지 스니펫이나 표 이미지가 패널에 표시됩니다. UI 상으로 문장마다 작은 **근거 아이콘** (서책 모양 등)을 두고, 이를 클릭하면 우측 패널이 해당 근거로 스크롤되어 하이라이트를 줍니다. 만약 근거가 여러 개 연결된 복합 문장이라면, 패널에 탭 또는 목록으로 모두 표시합니다 (예: ① 정책 문서 p5, ② 예산표 이미지 등). 이미지 근거의 경우, 썸네일과 함께 OCR 추출 텍스트를 나열하여 검색가능하게 하고, 필요하면 클릭해 확대합니다. 또한 패널 상단에 "출처 열기" 버튼으로 원본 파일을 새 창에 열 수 있게 합니다. 근거 패널은 편집과 독립적으로 스크롤 가능하지만, **연결된 모드**가 켜져 있을 땐 편집기에서 문장 이동 시 자동으로 해당 근거를 찾아줍니다. 예를 들어 문장 A에서 B로 커서 이동하면 패널도 B의 근거로 업데이트되는 식입니다.

  - **코멘트 스레드 (UI 하단):** 하단에는 전체 문서나 선택 문단에 대한 **댓글 영역**이 있습니다. Google Docs처럼 문장 범위를 지정해 댓글을 달기보다는, 각 문단의 액션들 중 " 댓글" 버튼을 눌렀을 때 하단에 해당 문단에 대한 쓰레드가 열리는 방식입니다. 댓글에는 사용자 아바타, 이름, 시간, 내용이 표시되고, **멤션(@)**을 지원하여 팀원 호출도 가능합니다. 특정 댓글에서 "해결됨" 체크를 하면 UI상 숨겨지거나 축소 표시해 토론 완료를 나타냅니다. 중요한 논의는 상단에 고정하거나 댓글 검색 기능도 부여할 수 있습니다. 모바일에선 이 댓글 패널을 화면 아래 시트(sheet) 형태로 띄워, 키보드로 입력하기 쉽게 전체화면으로 전환되도록 합니다.

  - **모바일 시나리오:** 모바일 웹에서는 기본적으로 읽기/승인 작업에 초점을 맞춥니다. 작은 화면에서 전체 편집은 어려우므로, **간소화된 인터페이스**를 적용합니다. 예를 들어 프로젝트 현장에서 실무자가 사진을 찍어 업로드하면, 모바일 UI에서 곧장 해당 사진과 자동 생성된 캡션이 표시됩니다. 실무자는 필요한 경우 캡션 텍스트를 편집하고 "제출"하면, PM은 데스크톱에서 그 항목을 확인하고 승인/수정하는 식입니다. 모바일에서는 문장별 승인/거부 토글이 더 크게 표시되고, 코멘트 달기도 한 화면에 바로 입력할 수 있게 합니다. AI가 문장을 자동 생성하는 과정은 서버에서 이뤄지고, 모바일에는 완료 시점에 푸시 알림(또는 웹 푸시/토스트)으로 알려주는 것도 고려합니다.

Implementation: 에디터 구현에는 **리치 텍스트 에디터(Rich Text Editor)** 라이브러리를 활용합니다. Draft.js, ProseMirror, TipTap 같은 도구에 우리의 커스텀 기능(diff, 승인 상태)을 얹습니다. 예를 들어 ProseMirror에 plugin을 만들어, 문장 단위 node를 추상화하고 AI 원본 텍스트를 comment로 저장해두면 diff 표시를 비교로 구현 가능합니다. 사용자가 편집을 마치고 포커스를 벗어날 때 해당 문장에 대한 diff를 계산하여, 삭제/추가 부분을 `<del>` `<ins>` 태그로 래핑하거나 span에 CSS로 빨간/초록 표시합니다. 또는 더 간단히, AI 생성문과 현재 편집문 두 버전을 모두 DOM에 두고 CSS로 하나를 strikeout/underline할 수도 있습니다. 승인/거부 상태는 문장 노드에 state 필드를 추가하거나, 별도 데이터 구조로 관리하면서 UI에 아이콘/배경색 변화를 줍니다. 예컨대 `<p data-status="approved">` 속성을 넣고 CSS로 배경을 연한 초록으로 할 수도 있습니다. **근거 연결**은 편집기 데이터 구조에 문장별로 source IDs를 매핑해두고, 사용자가 문장 클릭 시 그 ID로 백엔드에서 해당 부분을 가져오거나, 미리 로드해둔 검색 결과를 보여줍니다. 앞서 5)에서 구현한 검색/근거 뷰어 컴포넌트를 여기 재사용하거나, 편집기에 이미

AI가 사용한 근거 리스트를 알고 있다면 그것을 주입합니다. 이미지의 bounding box나 텍스트 anchoring은 PDF.js API 등을 활용하여 (x,y 좌표를 하이라이트) 구현할 수 있습니다 [30] [28]. **댓글**은 각 문단을 식별할 수 있는 ID (예: p1, p2 등)로 구분하여, 댓글 DB 테이블에 `paragraph_id`, `user`, `content`를 저장/조회합니다. Socket이나 realtime 기능으로 여러 사용자가 동시에 보면 새 댓글이 실시간 갱신되게 할 수 있습니다. 댓글 UI는 Draft.js의 단순 텍스트 에디터를 쓰거나, 멘션 `@` 입력 시 사용자 목록 검색 API를 호출해 autocomplete을 제공하는 식으로 구현합니다. 모바일의 사진 업로드는 `<input type=file accept="image/*" capture="camera">` 속성으로 카메라를 바로 열 수 있게 하며, 업로드 후 서버에서 Vision API나 OCR, Caption 모델을 통해 텍스트를 생성해 해당 프로젝트 로그에 새 문단으로 추가합니다. 그런 후 PM이 볼 땐 "(실무자)가 사진을 업로드함 - AI 캡션: ..." 형태로 표시되어 검토를 받게 합니다. 전반적으로, 멀티모달 에디터는 **협업 편집 시스템**에 가깝기 때문에, 충돌 관리(여러 사람이 동시에 같은 문단 수정 시)도 고려합니다. 이를 위해 문단 단위 editing lock을 걸거나, 마지막 저장자 기준으로 overwrite되되 이전 버전은 히스토리에 남기는 방식 등을 취합니다. UI 상에는 "실무자A가 편집 중..." 배지가 뜨도록 구현 가능합니다.

마지막으로, **일괄 편집 도구**(톤/용어 규칙 적용)는 사전에 정의한 치환 매핑이나 OpenAI API 등을 활용할 수 있습니다. 적용 시 비동기로 모든 문단을 훑으며 교정 제안을 하고, 각 변경은 diff로 표시하여 사용자 확인을 받습니다. 이러한 기능은 편집기 상단에 메뉴 버튼으로 제공하고, 실행 전 확인 모달("모든 문장에서 경어를 해체합니다. 계속하시겠습니까?")을 띄워 사용자 동의를 구합니다. 모든 편집이 완료되어 각 문단이 승인되면, "완료" 버튼을 눌러 다음 템플릿 매핑 단계로 넘어갈 수 있게 합니다 (결재 프로세스와 연결).

7. 템플릿 매핑 스튜디오 (HWP/PPTX)

Problem: 최종 문서(HWP 또는 PPTX)에 **AI 로그와 데이터**를 정확히 주입하려면, 템플릿 문서의 자리표시자(슬롯)를 데이터 필드와 연결(mapping)해야 합니다. HWP/PPTX는 각각 고유한 포맷(haansoft OLE, OOXML)으로 텍스트, 표, 이미지 위치 등을 정의하므로, UI에서 사용자가 복잡한 태그를 일일이 다루지 않고 **드래그&드롭** 등으로 직관적으로 매핑하도록 해야 합니다. 또한 **반복되는 영역** (예: 테이블 행 반복)이나 **조건부 섹션** (특정 조건 시 내용 포함/미포함)을 설정할 수 있어야 합니다. 이 단계에서 실수하면 문서 생성 실패나 잘못된 내용 삽입이 될 수 있으므로, UI 상에 **미리보기/검증** 기능이 필요합니다. 회사 정책상 **폰트**는 특정 폰트만 사용해야 하고, HWP/PPT 버전 호환성도 고려되어야 합니다. 템플릿 매핑은 일반 사용자가 생소할 수 있으므로, 사용성을 높이고 실수를 줄이는 것이 도전입니다.

Principle: UI에서 **양쪽 페인(left-right paradigm)**을 사용해 한쪽엔 **데이터 구조**(예: JSON 트리나 데이터베이스 필드 목록)를, 다른쪽엔 **템플릿 미리보기**를 보여줘 **소스-타겟 연결**을 쉽게 합니다. 이 패턴은 ETL 도구나 통합 플랫폼에서 흔히 쓰이며, 사용자가 **드래그 앤 드롭**으로 필드를 맵핑하거나, 클릭하여 매칭할 슬롯을 선택하는 인터랙션을 제공합니다. 필드 연결 시 타입이 맞지 않거나(예: 숫자를 텍스트필드에 넣는다든지) 하면 즉시 피드백(예: 경고 아이콘)을 제공해 오류를 예방합니다. Material3 등에서 강조하는 **"직관적 직접 조작"** 원칙을 따라, 사용자가 템플릿 문서의 특정 부분을 클릭하면 해당 부분에 연결된 데이터 필드를 지정하는 팝업이 뜨거나, 반대로 데이터 필드 옆에 현재 매핑된 템플릿 위치가 표시됩니다. **미리보기**를 통해 사용자는 실제 데이터가 들어갔을 때 어떻게 보일지 수시로 확인할 수 있어야 합니다. 또한, HWP/PPTX 포맷 제약으로 **폰트 대체**가 필요할 경우 (예: 템플릿에 회사폰트가 없으면 대체 폰트로) 그 사항을 사전에 명시해줘야 합니다.

Pattern:
- **좌측: 데이터 필드 패널:** 프로젝트의 데이터 스키마(JSON)나 데이터베이스 구조를 트리 형태로 보여줍니다. 최상위는 프로젝트나 보고서 객체, 그 아래로 섹션별 필드 그룹, 필드들이 노출됩니다. 필드명과 타입 아이콘(문자열, 숫자, 날짜, 이미지 등)을 함께 표시하고, 반복 가능한 리스트의 경우 [ ] 배열 아이콘으로 표시해 하위에 샘플 아이템 필드를 들여쓰기합니다. 예: `project -> basic_info -> title (text), date (date), members [ ] -> member_name, role, ...`. 이 패널에서는 필드 검색 기능(돋보기)을 제공하여 이름으로 필드 찾기가 가능합니다. 사용자는 이 패널에서 필드를 끌어서 오른쪽 **템플릿 캔버스**의 원하는 위치로 드롭할 수 있습니다.

    - **우측: 템플릿 캔버스:** 선택된 템플릿 HWP 또는 PPTX의 미리보기를 페이지별로 표시합니다. PPTX인 경우 슬라이드 썸네일 리스트를 좌측에 배치하고, 선택한 슬라이드를 크게 표시합니다. HWP인 경우 페이지 1, 2, ... 탭이나 스크롤로 표시합니다. 템플릿에 **필드 슬롯**이 미리 삽입되어 있다면 (예: `${project_name}` 같은 텍스

트 표식) 이를 UI가 감지하여 하이라이트 표시합니다. 또는 디자이너가 placeholder 텍스트에 특정 구분자를 넣는 방식이 아니라, **사용자 스스로 매핑 UI에서 지정**할 수도 있습니다. 예를 들어, 텍스트 상자를 클릭하면 "연결할 데이터 필드 선택" 드롭다운이 뜨고, 그 중 하나를 고르면 그 상자가 해당 필드로 매핑됩니다. 매핑된 슬롯은 특별한 마커(아이콘 or 테두리)로 표시하고, 클릭 시 어떤 필드와 연결됐는지 툴팁을 보여줍니다.

- **드래그 & 드롭 매핑:** 사용자가 좌측 필드를 우측 특정 영역으로 드래그하면, 그 영역(텍스트 상자, 표 셀 등)을 강조하여 "여기에 놓아서 연결" 가이드가 나타납니다. 드롭 시 해당 필드의 이름이나 placeholder가 그 위치에 삽입되고, 내부적으로 링크가 맵핑 테이블에 기록됩니다. 만약 대상이 **반복 영역**(예: 표의 행)이라면, 사용자가 배열 필드를 드래그할 때 표 전체에 드롭하면 한 행을 그 배열의 아이템 템플릿으로 지정합니다. 이후 UI에서 그 행은 노란색 배경 등으로 표시하고, 행을 복제 아이콘(∞)을 붙여 "반복됨"을 표시합니다. 조건부 섹션의 경우, 사용자가 특정 텍스트나 도형을 선택한 뒤 "표시 조건" 버튼을 누르면 해당 섹션에 논리 조건(예: 필드 X가 Y값인 경우 표시)을 설정할 수 있는 모달이 뜹니다. 거기서 필드와 비교 연산 등을 설정하면, 그 섹션은 연한 파랑 배경으로 표시되고 옆에 조건 아이콘(필터 모양)을 달아둡니다.

- **바인딩 규칙 편집:** 각 매핑된 필드에는 **포맷 옵션**이 있습니다. 필드 선택 시 화면 한쪽 (혹은 팝업)으로 속성 패널이 열리는데, 여기서 해당 필드의 표시형식을 조정합니다. 예: 날짜 필드는 `YYYY-MM-DD` 또는 `YYYY년 M월 D일` 등 포맷을 선택할 수 있고, 숫자 필드는 단위(만원, $ 등)나 소수점 자릿수, 천단위 콤마 여부 등을 설정할 수 있습니다. 또한 만약 필드 값이 없을 경우 대체할 **폴백 텍스트** ("N/A" 등)도 지정합니다. 이러한 설정은 JSON 스키마에서 타입 정보를 참고하여 제공됩니다. (ex. 날짜 타입인 경우에만 날짜 포맷 옵션 노출).

- **폰트 및 버전 관리:** UI 상단 또는 설정 패널에 "폰트 정책" 정보를 표시합니다. 예: "본 템플릿의 기본 글꼴: 맑은 고딕, 회사표준폰트: Pretendard, 생성 시 Pretendard로 대체 삽입 예정". 만약 템플릿이 회사 표준 폰트가 아닌 폰트를 사용 중이면 경고 아이콘과 함께 "이 폰트는 배포 PC에 없을 수 있습니다. 대체 폰트: Pretendard 적용" 안내를 합니다. (HWP의 경우 폰트 매핑 테이블 작성, PPTX의 경우 font fallback 설정). **버전 호환성**도 표시하여, 예를 들어 HWP 2020 버전 템플릿이라면 "생성 문서는 HWP 2020 양식으로 저장됩니다"라고 알립니다.

- **내역 및 협업:** 우상단에 "매핑 내역" 버튼을 두어, 매핑 스냅샷 리스트를 보여줍니다. 각 스냅샷에는 타임스탬프와 작성자, 변경사항 요약이 있고, 선택하면 해당 시점 매핑 구성이 로드되어 비교 확인 또는 **롤백**할 수 있습니다. 또한 이 스튜디오에서도 댓글/메모를 지원해, 예를 들어 "이 필드 데이터 확인 필요" 같은 메모를 템플릿 상 특정 요소에 남길 수 있습니다. 이는 템플릿 객체에 연결된 댓글로 구현됩니다.

Implementation: 이 스튜디오 UI는 **풀스크린 드로어** 형태로, React를 기반으로 구현됩니다. HWP는 웹에서 직접 렌더 어려우므로, 대안으로 PDF 변환한 미리보기 이미지를 사용하거나, pyhwpx를 활용해 HTML 표현을 생성하는 방법을 고려합니다. PPTX는 PptxGenJS 같은 라이브러리로 JSON 파싱하거나, Office365 프레젠테이션 Embedding을 쓰는 방법도 있지만 오프라인 환경이라 가정하면 제한됩니다. 현실적으로, **슬라이드/페이지 미리보기**는 서버에서 png로 생성해 전달하고 (예: 1페이지마다 png), 클라이언트는 그 이미지를 캔버스나 img 태그로 배경 표시합니다. 그리고 텍스트 영역 클릭 시 해당 위치(좌표)에 가장 가까운 텍스트 상자를 식별하기 위해, PPTX의 placeholder 정보를 사전에 추출해둘 수 있습니다 (python-pptx 이용). python-pptx를 통해 placeholder나 텍스트 상자의 idx, 내용, 좌표를 추출하고, 이를 JSON으로 보내주면, 프론트는 해당 위치에 보이는 투명 레이어 DIV를 배치해 클릭 핸들링합니다. (31 플레이스홀더 활용하면 슬라이드에 미리 정의된 상자 활용 가능 32). HWP의 경우도 pyhwpx로 hwpml 파싱해 위치를 얻거나, 편의상 HWP는 한글 swf 도구나 PDF 경유로 처리할 수 있습니다. **드래그&드롭**: HTML Drag and Drop API로 구현, 필드 패널 아이템에 `draggable=true` 속성을 주고, 캔버스의 drop zone을 페이지 전체로 받아 드롭 좌표를 계산합니다. 드롭 시, 해당 좌표와 내부 매핑 데이터(placeholder list 등)로 가장 가까운/해당하는 템플릿 요소를 결정하여 필드 매핑을 수행합니다. 만약 사용자 입력으로 직접 placeholder를 찍고 싶다면, "+ 새 필드 삽입" 메뉴를 템플릿에 제공하여, 이를 누르고 페이지 클릭하면 새 텍스트 상자를 그 위치에 생성하고 편집 가능하게 (특정 지원 어려우면 차선책: placeholder 텍스트를 나중에 치환). **매핑 저장**: 매핑 정의는 JSON으로 유지됩니다 (예: `{"field": "basic_info.title", "target": {"slide":1,"shape":3}, "format": "upper", ...}` 등). 저장 버튼 누르면 이 JSON을 백엔드에 저장, 문서 생성 시 참고하게 합니다. **반복 영역**: UI에서 표를 지정하면 JSON에 `repeat:` 필드로 표시하고, 생성 코드에서 해당 표 행을 템플릿 행 복사 후 데이터 반복

채우도록 합니다. (python-pptx에는 table row 추가 API가 있고, pyhwpx는 표 셀 XML 복제 가능). **조건부**: JSON에 조건식 (예: `condition: "basic_info.budget > 0"`)을 넣고, 생성 시 파싱하여 false면 그 요소를 삭제. OOXML(PPTX)의 경우 shape 제거, HWP는 한글 컨트롤 명령(숨김글) 같은 걸 활용하거나, pyhwpx로 dom 수정. **폰트 처리**: pyhwpx, python-pptx로 텍스트 넣을 때 폰트 설정 가능하므로, 매핑 시 지정한 대체 폰트를 일괄 적용. e.g., python-pptx에서 `shape.text_frame.paragraphs[0].font.name = "Pretendard"` 등 처리.

미리보기 & Diff: 사용자가 "미리보기" 버튼을 누르면 현재 매핑을 적용한 결과 문서를 **실시간 생성**하여 보여줍니다. 이는 서버에서 실제 데이터를 넣어 pdf나 이미지로 반환하는 방식일 수 있습니다 (시간 오래 걸리므로 진행 표시 필요). 또는, 프론트에서 미리보기 모드로 필드값을 직접 치환해볼 수 있습니다. 예를 들어 JSON 데이터가 존재한다면, 그 값을 캔버스의 placeholder 텍스트 자리에 실시간으로 넣어보여 주는 것인데, 이미지 기반 미리보기에서는 어려워서, 차라리 **로컬 PDF 다운로드** 버튼으로 대체할 수도 있습니다. Diff 기능은 최종 산출 문서와 이전 버전을 비교하여 달라진 문단에 하이라이트 하는 것으로, 구현상 난이도가 높으므로, v1에서는 일단 **텍스트 기반** 비교로 간소화 (예: 보고서의 텍스트 추출하여 이전 텍스트와 비교, 변경된 문단을 리스트업).

협업/버전: 매핑 편집 내역은 중요한 변경(새 필드 연결/반복/조건 설정 등) 시점에 JSON 스냅샷을 백엔드에 기록하여 관리합니다. UI에서 이를 불러와 과거 JSON과 현재 JSON diff를 보여줄 수 있고(텍스트 diff 라이브러리 활용), 롤백 선택 시 현재 JSON을 그 버전으로 교체하고 사용자에게 확인을 받습니다. 댓글 기능은 Figma의 주석처럼 특정 요소를 지정하기 복잡하면 화면 전체 코멘트로 처리하거나, "전체 템플릿"에 대한 논의로 간주할 수 있습니다.

결과적으로, 템플릿 매핑 스튜디오는 **시각적인 매핑 편집기**와 **데이터 바인딩 설정 폼**이 결합된 형태이며, 사용자가 코딩 없이도 문서 자동화 룰을 완성하도록 돕습니다. UI 상에서 최대한 오류를 방지하고 (불일치 타입 경고 등), HWP/PPTX 포맷 한계는 백엔드에서 제어하는 방식으로 역할을 분담합니다.

8. 미리보기 · 결재 · 내보내기

Problem: 모든 편집과 매핑이 끝난 후 사용자는 최종 산출물을 **검토(미리보기)**하고, 내부 프로세스에 따라 **결재 승인**을 얻어야 하며, 그 후 대외적으로 문서를 **배포(내보내기)**해야 합니다. 이 단계에서 발생할 수 있는 문제는, 최종 문서가 기대와 다르게 생성되었거나(예: 레이아웃 깨짐, 오타), 결재자가 수정 요청을 하는 경우 워크플로우 처리, 그리고 문서 내보낼 때 **보안**을 유지하는 것입니다. 또한 HWP/PPTX라는 **바이너리 포맷** 특성상 웹에서 바로 렌더링이 어려워 미리보기를 효율적으로 제공하는 방법도 고민됩니다. 내보내기 시 만료 링크나 비밀번호, 워터마크 등을 설정할 수 있게 하여 문서 유출 위험을 줄이는 것도 필요합니다.

Principle: WYSIWYG(보는 대로 출력)을 최대한 보장하는 미리보기를 제공합니다. 사용자가 최종 파일을 열어보지 않고도 브라우저에서 모든 페이지를 확인할 수 있어야 하며, 이전 버전과 달라진 부분을 알 수 있으면 좋습니다. 결재 프로세스는 **명확한 단계와 책임자 표시**를 원칙으로 합니다. 전자결재 UI는 각 결재자(또는 역할)의 승인/반려를 기록하고, 가능한 간단한 인터랙션 (원탭 승인, 코멘트 입력 등)으로 진행되어야 합니다. 최종 내보내기는 다양한 방법(PPTX/HWP 파일 다운로드, 이메일 발송, 사내 저장소 업로드 등)을 지원하되, **접근 통제** 기능을 옵션으로 제공합니다. 예를 들어 다운로드 링크 만료 기간, 비밀번호 설정, 워터마크 삽입을 사용자에게 설정하도록 함으로써 **보안**과 **편의**의 균형을 유지합니다.

Pattern:
- **문서 미리보기:** 최종 문서(HWP/PPTX)를 페이지 이미지로 변환하여 앱 내에서 보여줍니다. PPTX의 경우 슬라이드별 썸네일을 좌측에 리스트로 두고, 클릭 시 우측에 큰 슬라이드 이미지를 표시합니다. HWP는 페이지 번호 탭이나 세로 스크롤로 페이지 이미지를 표시합니다. **변경 내용 하이라이트:** 만약 직전 버전 대비 업데이트된 보고서라면, 변경된 문단이나 표에 노란 형광펜 배경을 깔아 보여줍니다. (이 diff 정보를 얻으려면 비교 기능이 필요하지만, 간단히는 AI 로그 생성 단계에서 수정된 문단에 표시했던 것을 활용하거나, 버전간 텍스트 비교로 가능). **확대/축소** 기능을 제공하여 세부 확

인이 가능하도록 하고, 전체 화면 보기 옵션도 둡니다. 각각의 페이지는 캔버스나 img 태그로 표시되며, 텍스트 검색(문서 내 find) 기능이 있다면 OCR된 텍스트나 pdf텍스트를 기반으로 구현 가능합니다.

- **결재 (Approval) UI:** 결재선에 지정된 사람들에게 문서를 승인을 요청하는 흐름입니다. UI 상에서 "결재 요청 보내기" 버튼을 누르면, 지정된 승인자들에게 알림이 가고 상태가 "승인 대기"로 표시됩니다. 결재를 위한 별도 모달이나 페이지를 제공하여, 결재자는 문서를 검토한 후 "승인" 또는 "반려" 버튼을 누를 수 있습니다. **결재선 표시:** 대시보드나 문서 화면에 현재 문서의 결재 진행 상태를 시각적으로 표시합니다. 예: 직급/이름과 함께 각자 상태(승인됨 ✓, 반려 ✕, 대기 …)를 아이콘으로 표기하고, 순차 결재라면 순서대로 진행됩니다. 결재자가 코멘트를 남길 수 있게 하여, 반려 시에는 반드시 사유를 입력하도록 합니다. 전자서명: 승인 시 본인 인증을 위해 암호 재입력이나 2FA를 요구할 수 있고, **전자서명 이미지**(스캔 사인) 등록이 되어 있다면 문서에 삽입되게 할 수도 있습니다. UI에서는 결재자가 "서명 삽입" 옵션을 체크하면, 시스템이 최종 문서 PDF에 서명란 이미지를 넣는 백엔드 처리를 합니다. **결재 완료 후**에는 문서 상태를 "승인 완료"로 표시하고, 내보내기 단계로 넘어갈 수 있게 Unlock합니다.

- **내보내기 옵션:** 사용자가 최종 산출물을 다른 채널로 배포하는 기능입니다. **파일 다운로드:** 단순히 PPTX/HWP 파일을 다운로드 링크로 제공하며, 클릭 시 다운로드됩니다. 이때 버전명을 파일명에 넣어 ("Report_v1.0_승인됨.pptx") 관리에 도움을 줍니다. **이메일 전송:** UI에서 이메일 입력 혹은 사전 정의된 수신자 리스트를 선택하고 "전송"하면, 해당 파일을 첨부하거나 링크를 포함한 메일을 발송합니다. 수신자에게는 만료 기간 및 비밀번호 정보도 전달되어야 합니다. **만료 링크 생성:** "만료 공유 링크" 기능을 통해 문서를 임시로 호스팅하고 URL을 발급합니다. UI에서 만료기간 (예: 7일, 30일, 커스텀), 다운로드 허용 여부, 접근 비밀번호를 설정할 수 있는 폼을 제공합니다. 생성 버튼 클릭 시 링크 URL과 QR코드를 보여주고, 복사 기능을 제공합니다. 수신자가 링크를 열면, 서버에서 권한 체크 후 다운로드 페이지를 표시합니다. **워터마크 옵션:** 내보내기 전에 문서에 보안 워터마크(예: "Confidential - Company Name - User Email - Timestamp")를 추가할 것인지 체크박스로 선택하게 합니다. HWP/PPTX에 워터마크 삽입은 백엔드에서 처리하므로, UI에서는 옵션만 주고 설명툴팁을 제공합니다 ("문서 모든 페이지에 열람자 정보 워터마크 삽입"). 외부 공유 시엔 켜도록 기본 체크.

- **결재·배포 로그:** 결재나 내보내기 행위도 **감사 로그**에 기록되므로, UI에서 "내보내기 완료 - 링크 생성됨 (만료 2024-01-01)" 같은 메시지를 남깁니다. 또한 내보낸 후 "(클릭 수 X회, 다운로드 Y회)" 등의 통계가 필요하면, 링크 관제 시스템과 연계하여 UI에 표시할 수 있습니다.

Implementation: 미리보기는 **서버사이드 문서 → PDF/PNG 변환**을 활용합니다. 예를 들어 PPTX→PNG 라이브러리 또는 LibreOffice headless convert를 사용해 전체 페이지 이미지를 생성해 프론트에 전달합니다. HWP는 OLE automation이나 서버 전용 API (한글 웹 오피스 API 등)로 PDF를 만들 수 있다면 활용하고, 어려우면 pyhwpx로 HTML 얻어 프론트에서 스타일 적용하는 차선도 고려합니다. 프론트에서는 받은 이미지를 `<img>` 태그 목록으로 표시하며, lazy loading을 적용해 초기 로드 성능을 높입니다. Diff 하이라이트는, 두 버전 문서의 텍스트를 추출해 비교하는 서버 기능을 호출하거나, 버전 간 AI 로그의 diff로 대략 대체할 수 있습니다. 결재 프로세스: 결재선과 상태는 데이터베이스에 모델링되어 있습니다 (예: approvals 테이블에 문서ID, 결재자, 순서, 상태, 코멘트). 프론트는 이를 불러와 UI에 표시하고, 결재자가 승인 버튼 누르면 PUT/PATCH 요청으로 상태를 업데이트합니다. 승인/반려 액션 시 보안 상 재확인이 필요하면, UI에서 비밀번호 입력 모달을 띄우고 서버에 검증 요청을 하는 식으로 합니다. (또는 SSO 세션 기반이면 생략 가능). 결재 알림은 이메일이나 사내 메신저 연동할 수 있으나, 우선 앱 내 알림(벨 아이콘에 뱃지)으로 구현합니다. 전자서명 이미지는 사전에 사용자마다 업로드받아 저장해두고, 승인 시 서버에서 docx나 PDF에 그 이미지를 삽입 (예: PDFtk 등으로). UI에선 이미지 미리보기나 업데이트 기능을 사용자 프로필 설정에 제공합니다.

내보내기: 파일 다운로드는 단순 링크 (Auth 필요한 경우 일회용 토큰 링크)로 구현합니다. 이메일 전송은 서버에서 SMTP or 이메일 API 활용. 공유 링크 생성은 DB에 token, 만료일, 비번 hash 저장하고 프론트에 URL 구성. 만료 기간이 지나면 다운로드 API가 410 Gone 오류를 리턴하도록 합니다. UI에서 비밀번호 설정하면, 수신자는 링크 클릭 시 비번 입력 페이지가 나오고 맞으면 다운로드 시작됩니다. 워터마크: 서버에서 PPTX에 투명 텍스트박스, HWP에 배경 텍스트 추가하는 코드 실행. 이러한 옵션 적용 여부는 API 파라미터로 보내 처리합니다.

마지막으로 UI 피드백: 내보내기 완료 후 “문서가 성공적으로 배포되었습니다” 토스트, 또는 이메일 전송 결과(성공/실패) 피드백 제공. 결재 완료 후에도 “모든 결재가 완료되었습니다” 등의 안내를 줘 사용자 다음 단계를 명확히 합니다.

9. 권한 · 설정 · 감사로그

Problem: 기업용 시스템이므로 **다양한 권한 레벨**과 세밀한 보안 설정이 필요합니다. 프로젝트마다 접근 권한이 다를 수 있고 (RBAC), 외부 감사 계정은 오직 조회만 가능해야 합니다. 또한 시스템 전반 설정(UI 옵션, API키 관리, PII 마스킹 규칙 등) 인터페이스가 필요합니다. **감사로그**는 누가 언제 무엇을 했는지 추적하는 기록으로, 이를 관리자가 조회하고 필터링할 UI가 필요합니다. 이 모듈에서 중요한 점은 **보안** – 권한 없는 사용자가 설정이나 로그를 볼 수 없어야 하며, UI 또한 방대한 로그를 효율적으로 보여줘야 한다는 것입니다. 1년 기본 보존, 3~5년 옵션 보존 등의 정책도 UI에 반영되어야 합니다.

Principle: 최소 권한 & 명시적 제어를 따른 UI 설계가 필요합니다. 권한 설정 UI에서는 프로젝트별로 어떤 사용자가 어떤 역할인지 한눈에 파악하고 변경할 수 있어야 합니다. Role 기반 권한 정책은 사전에 정의되어 있으므로, UI에서 **직관적인 스위치/체크박스**로 할당합니다 (예: 사용자 목록에서 Role 드롭다운 변경). 중요한 설정 (예: API 키 발행, 데이터 보존기간 변경)은 **재확인(confirm)**을 거치고, 활동 로그에 남겨야 합니다. **감사로그 UI**는 방대한 로그를 다루므로 **필터와 검색**이 핵심입니다 – 날짜 범위, 사용자, 이벤트 종류 등을 입력받아 필요한 로그만 보여줘야 합니다 [33]. 또, 로그는 쉽게 **내보내기(CSV)** 할 수 있게 하여 감사 목적으로 외부 제출도 지원합니다. 접근성 면에서도 표 형태의 로그를 잘 구조화하고, 긴 리스트는 가상 스크롤로 성능을 유지합니다.

Pattern:
- **프로젝트 권한 관리:** 관리자나 PM은 프로젝트 설정 화면에서 멤버 리스트와 그들의 역할을 관리합니다. UI에서 프로젝트 멤버 섹션을 만들고, 각 멤버 행에 이름, 이메일, 현재 역할(Role)을 보여줍니다. 역할 컬럼은 드롭다운 or Pill로 현재 역할 표시하고, 클릭 시 변경 가능한 역할 목록(Admin/PM/Editor/Viewer 등)을 제시합니다. 변경 시 바로 적용되며 (또는 “저장” 버튼으로 일괄 저장), 변경되면 해당 사용자에게 알림을 보낼 수도 있습니다. 신규 멤버 추가는 상단 “멤버 초대” 버튼으로 이메일 입력 & 역할 선택해서 invite 합니다. 외부 감사 계정을 추가할 때는 조회전용 역할만 선택 가능하도록 제한하거나, UI에서 “외부감사 초대” 별도 플로우로 표시하여 구분할 수 있습니다.

  - **필드 단위 권한:** 특정 민감 필드는 일반 사용자에겐 마스킹된 값만 보여줘야 한다면, UI에서 해당 필드에 “민감/비공개” 토글을 표시해 관리자가 설정할 수 있습니다. 예컨대 개인정보 항목(전화번호 등)에 자물쇠 아이콘 토글이 있고, 켜면 실무자에겐 `***-****`처럼 마스킹되게 처리합니다. 이러한 UI 요소는 데이터 스키마와 연계되어, 토글 on 시 RLS 정책이 해당 필드에 적용됩니다.

  - **설정 (Settings):** 사용자 프로필 및 시스템 설정 메뉴를 둡니다. 프로필에서는 비밀번호 변경, 다중 인증 설정, 전자서명 업로드 등을 제공하고, 시스템 설정(관리자 전용)에서는 API 키 관리, PII 규칙, 데이터 보존, 테마 등 옵션을 다룹니다. **API 키 보관:** API 키(예: 오픈AI 키 등)를 입력받아 암호화 저장한다는 안내와 함께 입력 폼을 제공하고, 입력 시 "저장" 누르면 키 일부만 마스킹된 채로 확인 표시합니다 (사용자에게 키 원문은 다시 안보이게). **PII 마스킹 규칙:** 여러 패턴(주민번호, 이메일 등)에 대한 마스킹 정책을 ON/OFF하거나 정규식 편집 UI를 제공합니다. 예: "주민등록번호: ●●●●●●-●●●●●●●" 식으로 미리보기 예시와 함께 ON되면 업로드 시 자동 마스킹됨을 설명합니다. **보존 기간:** 드롭다운으로 1년, 3년, 5년 선택하게 하고, 변경 시 “이 설정은 모든 기존 로그 삭제 일정에 영향을 줍니다. 변경하시겠습니까?” 컨펌. 각 설정 항목에는 “기본값 1년” 같은 라벨도 명시합니다.

  - **감사 로그 화면:** 관리자 메뉴에서 접근. 화면 상단에 **필터 바**를 제공합니다. 여기서 기간(오늘, 7일, 기간 지정), 사용자 (드롭다운 or 검색), 이벤트 종류 체크박스 (보기/다운로드/편집/승인/내보내기 등 주요 카테고리) 등을 선택 후 “필터 적용” 버튼을 누르면 조건에 맞는 로그만 표시합니다 [33]. 로그는 표 형태로 Timestamp, User, Action, Target, Detail 등의 열이 있습니다. 예: `2025-01-05 14:22 | userA | VIEW | Document XYZ | (IP, location)` 정도. 너무 긴 상세는 툴팁이나 펼침 화살표로 숨기고, 한 행 클릭하면 우측에 상세 패널이나 하단에 확장되어 전체 내용을 보여줍니다. 로그는 시간순 정렬(default 최신순)이며, 페이지당 50개 또는 가상스크롤. **내보내기:** 필터링된 결과

를 CSV/PDF로 추출하는 버튼을 상단에 둬, 클릭 시 다운로드합니다. **보존정책 표시:** 로그 화면에 "로그 보존 기간: 1년 (2024-01-01 이후 자동 삭제)" 같은 안내를 상단에 배너 또는 작은 글씨로 표시합니다.

- **보안 세션:** 설정 중 **세션 관리**가 있다면, 현재 로그인 중인 브라우저/기기 리스트를 보여주고 "로그아웃" 시킬 수 있게 하는 UI (일반 계정 설정). 또는 **RLS 정책 미리보기**: admin이 특정 사용자나 역할을 선택하면 그 권한으로 보이는 메뉴를 미리볼 수 있는 "가상 세션" 같은 기능을 제공하면 좋지만, 필수는 아닙니다.

Implementation: 권한에 따라 UI 요소를 표시/비활성화하는 것은 프론트에서도 적용합니다. 예를 들어 Viewer(외부 감사)로 로그인하면 사이드바 메뉴에서 설정/로그 등 admin 메뉴를 렌더하지 않습니다. 이 정보는 Auth JWT나 세션 정보에 포함된 role로 분기합니다. 하지만 **신뢰성**을 위해 백엔드 RLS가 궁극적으로 데이터 액세스를 통제하므로, 프론트에서의 숨김은 UX 편의입니다 [1] . 프로젝트 멤버 관리 UI는 프로젝트 API (members list, add member 등)를 호출하여 변경을 수행합니다. 역할 변경 시 PUT 요청으로 서버에서 권한 업데이트 & 해당 유저에게 이메일 알림 (예: "당신은 프로젝트 X의 Editor로 지정됨"). 필드 단위 마스킹 토글은 서버의 메타데이터(API) 업데이트; RLS 정책은 DB pgpolicy에 저장되므로, admin UI 조작 -> RPC로 해당 policy toggling. 설정 화면은 다양한 항목이지만, 구조화하면 REST/GraphQL 등의 설정 엔드포인트에 POST/GET을 보내는 식입니다. API 키 입력은 프론트에서 평문 받아 즉시 https로 전송 -> 서버에서 암호화 저장. 프론트엔드는 재조회 시 마스킹 처리된 문자열만 받게 합니다 (예: "sk-***1234").

감사로그 UI: 서버로부터 로그 데이터를 페이지네이션/스트리밍 방식으로 받아옵니다. 한번에 너무 많은 양을 불러오지 않도록, 기간 필터 등으로 제한 요구. 또, 로그 DB는 인덱싱이 되어있어야 조회 빠르므로 UI에서도 최소 1일 이내 등 기본 필터를 적용합니다. 필터/검색을 수행하면 쿼리 파라미터를 API에 전달해 해당 조건의 로그만 JSON으로 가져옵니다. UI Bakery 사례처럼 **시간, 사용자, 이벤트 종류별 필터**는 UI에서 많이 쓰는 패턴입니다 [33] . UI에서는 DataTable로 로그 표시하며, virtualization 적용 (React Window 등). 내보내기 CSV는 그냥 API에서 동적으로 생성된 CSV를 blob으로 내려주거나, 프론트에서 JSON을 csv-string으로 변환도 가능합니다 (대량일 경우 메모리 문제 고려).

감사로그 항목 클릭 시 더 상세한 정보(예: 객체 변경 전후 값, 에러 스택 등)가 있다면 표시하는데, 디자인적으로 오른쪽에 Drawer로 "로그 상세" 보여주는 방법이 깔끔합니다 [34] (HighLevel 같은 서비스는 로그 클릭 -> 우측 Drawer에 세부정보 패턴 [35]).

Retention: 보존 기간 설정이 변경되면, 서버 스케줄러가 오래된 로그를 지우도록 할 것입니다. UI에서는 그 정책을 표시만 하므로, 보존기간 값을 가져와 표기. 3년 등 옵션 시, admin이 UI에서 기간 선택 -> Confirm -> 서버 설정 update.

마지막으로, **Audit trail of audit changes**: 감사로그에도 누가 설정 바꿨는지 들어가야 합니다. 예컨대 admin이 보존기간 1년->3년 변경하면 그 액션도 로그에 기록. 그런 meta 변경도 events category로 분류 (Settings 변경).

전반적으로, 권한/설정/로그 UI는 **정보량이 많고 복잡**하지만, 논리적 그룹과 직관적 컨트롤(토글, 드롭다운, 필터)을 사용하여 사용성이 높도록 설계합니다.

10. 접근성 · i18n · 모바일

Problem: 다양한 사용자가 (장애 포함) 사용할 수 있도록 WCAG 2.2 AA 기준을 충족해야 합니다. 또한 초기에는 한국어 UI지만 다국어 (영어 등)로 확장할 국제화(i18n) 준비가 필요합니다. 모바일 사용 시 주요 기능(사진 업로드, 댓글, 승인 등)을 지원하되, 화면 크기 제약으로 전체 기능을 모두 동일하게 보여주기는 어렵습니다. 따라서 **모바일에서의 축소 경험**을 정의해야 합니다. 접근성 면에서는 색상 대비, 키보드 조작, 포커스 표식, 폰트 크기 등 세부 지침을 전부 따르는지 확인해야 합니다. i18n은 날짜/시간/숫자 포맷의 현지화와 UI 문자열 번역 준비를 의미합니다.

Principle: 포괄적 디자인 원칙 아래, 비장애인뿐 아니라 스크린리더 사용자, 색각 이상자, 저대조 환경 등 누구나 동등한 정보를 얻도록 UI를 구성합니다. 이는 Perceivable, Operable, Understandable, Robust (POUR)라는 WCAG의

4대 원칙을 토대로, **대체 텍스트**, **키보드 내비게이션**, **충분한 명도대비** 등을 구현하는 것입니다 [36] [37]. 예컨대 모든 아이콘에는 `aria-label` 또는 화면리더용 텍스트를 제공하고, 의미 색상에는 아이콘/텍스트로 보완합니다 [15]. 포커스가 어느 UI 요소에 있는지 항상 시각적으로 표시하고, 포커스 링은 2px 두께 이상, 3:1 대비색으로 눈에 띄게 합니다 [38]. 터치 타겟은 WCAG 2.5.8에 따라 **최소 24x24px** 크기로 하며, 인접 타겟과 24px 간격을 유지해 잘못 누르지 않게 합니다 [12]. i18n은 UI 구조와 콘텐츠를 **분리**하고, 모든 문자열을 한 곳에서 번역 가능하도록 처리합니다. 형식(날짜 등)은 사용자 로케일에 맞게 (한국어=YYYY-MM-DD, 영어=MM/DD/YYYY 등) 표시되도록 국제화 라이브러리를 사용합니다. 모바일에서는 **핵심 작업 우선**으로 UX를 단순화합니다. 작은 화면에 전체 UI를 축소하면 사용성이 떨어지므로, 업로드/코멘트/승인 등 중요한 기능은 **하단 고정 바**나 부각된 버튼으로 배치해 원터치로 실행되게 합니다.

Pattern:
- **명도 대비 & 색상:** 텍스트와 배경색 대비비는 4.5:1 이상 (큰 텍스트 3:1)으로 유지 [39]. 모든 UI 컴포넌트의 색 조합을 점검해, 에러 빨강 vs 흰 배경, 링크 파랑 vs 흰 배경 등이 기준을 충족하도록 디자인합니다. 색약 사용자를 위해 **색상만으로 정보 전달 금지**: 예를 들어 그래프에서도 색 구분 외에 패턴이나 레이블을 추가, 필수 입력도 빨간 테두리 뿐만 아니라 "(필수)" 텍스트 병기. **포커스 표시:** CSS `:focus-visible` 스타일을 이용해 키보드 포커스 요소에 두드러진 outline을 제공하며, 일반 마우스 클릭 시는 표시 안 해 혼란 감소. 포커스 링 컬러는 보조 색 (예: Primary Light #93C5FD)이나 검정으로 하고, 두께 2px, 혹은 배경 하이라이트를 사용할 수도 있습니다 [38]. **키보드 순서:** DOM 순서가 논리적 흐름과 일치하게 컴포넌트 배치하고, tabindex를 남용하지 않으며, 모달 열리면 그 내부 첫 요소에 포커스 이동, 닫으면 이전 트리거로 포커스 복귀. **ARIA 레이블:** 아이콘 버튼 (예: 검색, 닫기 등)에 aria-label 속성으로 용도를 명시합니다. 실시간 업데이트 영역(예: 업로드 진행 리스트)은 aria-live를 활용해 변화를 알립니다. **모션 감소:** CSS 애니메이션/트랜지션에 `prefers-reduced-motion: reduce` 미디어쿼리를 적용해, 사용자가 OS에서 모션감소 설정 시 부드러운 fade정도로만 전환되게 합니다. 큰 움직임(슬라이드 애니 등)은 disabled.

- **국제화(i18n):** 모든 UI 텍스트 (라벨, 버튼, 메시지 등)를 코드에 하드코딩하지 않고 별도 리소스 파일(locale JSON 등)에서 가져오도록 합니다. 기본 한국어 (`ko.json`)로 키-값을 만들고, 이후 영어(`en.json`)를 추가하는 구조입니다. 예: `{"UPLOAD": "업로드", "UPLOAD": {"ko": "업로드", "en": "Upload"}}` 형태. UI에서 문자열 연결 시도는 지양하고 템플릿 방식(`$1님이 $2를 업로드했습니다`)으로 작성해 다른 언어 어순을 대처합니다. 날짜/시간 포맷도 i18n 라이브러리(e.g., date-fns, Intl API)를 사용해 현지화. 한국어: YYYY-MM-DD 24시간제, 영어: MM/DD/YYYY 12-hour AM/PM 등. 통화/숫자도 `toLocaleString(locale)` 활용. **RTL 지원:** 잠재적으로 오른쪽에서 왼쪽 읽기 언어도 고려해, CSS에서 text-align이나 margin-left 등 대신 logical properties(`start`/`end`) 사용을 권장하고, 아이콘과 텍스트 순서가 뒤바뀌어도 어색하지 않게 구조분리. (당장은 아닐지라도 염두).

- **모바일 디자인: 반응형 레이아웃**을 정의합니다. 브레이크포인트 예: <640px 모바일, 640~1024 태블릿, >1024 데스크톱. 모바일에서는 좌측 사이드바 내비게이션을 **햄버거 메뉴**로 숨겨두고, 상단에만 로고/페이지제목/햄버거 아이콘 표시합니다. 주요 알림은 상단 별도 벨 아이콘으로 대신. **핵심 시나리오 3가지** – 사진 업로드, 댓글, 승인 – 를 빠르게 할 수 있도록, 예컨대 모바일 홈화면에 "사진 업로드" 큰 버튼을 배치하여 누르면 프로젝트 선택 및 카메라 호출 흐름으로 바로 진입하도록 합니다. 댓글/승인은 알림을 통해 접근할 수 있게, 모바일 푸시(web push)나 SMS 등도 고려. **모바일 상호작용**: 드래그 앤 드롭 등 데스크톱 제스처는 모바일에서 어려우므로, 업로드는 탭하여 파일선택, 템플릿 매핑에서 드래그 대신 필드 선택 후 "맵핑할 위치 지정" 모드로 클릭 등 대안 제공. 표나 그래프는 가로 스크롤로 overflow 가능하게. **터치 영역**은 손가락 고려해 버튼 크기 크게, 간격 넉넉히. 폰트는 iOS/안드로이드 시스템 폰트 우선 사용.

- **튜토리얼 및 안내:** 접근성 측면에서, 처음 사용하는 사용자를 위한 온보딩 안내를 단순 텍스트뿐 아니라 **동영상/음성 대본** 등 다양한 형태로 제공해 각자 편한 방식으로 학습하게 합니다. 예컨대 첫 업로드 시점에 툴팁 투어 + 도움말 링크(스크린리더용 숨김 설명 포함)를 제공하면 좋습니다.

Implementation: 접근성 테스트를 위해 스크린리더 (NVDA/VoiceOver)로 주요 흐름을 점검하고, 수동 체크리스트를 완료합니다. 예를 들어: 모든 이미지에 alt가 있는지 (장식적 이미지 alt="" 처리), 제목 계층이 논리적인지 (`h1` 하나, `h2/h3` 순차), 양식 레이블이 연결되었는지 (`<label for>` 또는 aria-label), 키보드로 모든 인터랙션이 가능한지(Tab, Space/Enter로 버튼 활성 등), 초점이 초기로 가지 않게 하는지 (focus management) 등을 확인합니다. 색

상 대비는 Figma 플러그인이나 Chrome 확장으로 점검하고, 4.5:1 미만 나오면 색 어둡게 조절합니다 [39]. 포커스 링 구현은 Tailwind 설정에 `ring-offset` 등을 추가해 시각적 스타일을 통일하고, CSS에서 `:focus-visible` 사용.

i18n은 React Intl 혹은 i18next를 적용하여, `<Text id="upload" />`처럼 사용하며, 앱 상태에 `locale`을 두고 Phase 2에 영문 리소스만 추가하면 바로 전환되게 준비합니다. 모든 문자열 리소스는 영문도 채워넣어 (개발 편의), 한국어 UI 시 기본 `ko`를 로드합니다. (한국어 특유의 단위/포맷은 Intl API locale=ko-KR로 처리).

모바일은 CSS 미디어쿼리와 적응형 컴포넌트로 구현합니다. 사이드바는 작은 화면에서 `<Drawer>`로 구현하고, 햄버거 버튼 (`aria-label="메뉴 열기"`) 클릭 시 `sidebar.open=true`. 주요 버튼들은 `fixed bottom-0` 바 등에 배치하여 항상 노출. 예를 들어 `<button class="fixed bottom-4 right-4 bg-primary-600 text-white p-4 rounded-full shadow-lg">사진 업로드</button>` 등 플로팅 액션 버튼 패턴.

또한 모바일 성능 위해 불필요한 애니메이션 제거, 이미지/캔버스 최적화, 터치 지연 제거 (CSS `touch-action` 등) 등도 적용합니다. 마지막으로, **테스트**: 다양한 화면 크기(devices)와 스크린리더, 고대비 모드, 색약 시뮬레이션 등을 통과하도록 QA를 수행합니다.

11. 상태 (로딩/빈화면/에러) 패턴

Problem: 데이터가 로드되기 전에 나타나는 **로딩 상태**, 콘텐츠가 없을 때의 **빈(empty) 상태**, 그리고 오류 발생 시의 **에러 상태**를 빈틈없이 디자인해야 합니다. 사용자에게 아무 표시도 없이 기다리게 하거나, 에러를 이해 못하게 하면 UX 품질이 떨어집니다. 특히 이 시스템은 긴 작업이 많아 (분 단위 로딩) 사용자 인내심을 관리해야 합니다. 또한 작업이 백그라운드로 진행되는 경우 알림을 주고, 누락 없이 로그도 남겨야 합니다.

Principle: Skeleton vs Spinner 기준: 앞에서 논의했듯, **짧은 로딩**(2초 이내)은 단순 Spinner로 처리할 수 있지만, **긴 로딩**(여러 초 이상)은 Skeleton이나 progress bar로 현재 진행을 보여주는 것이 좋습니다 [24]. 항상 **시스템 상태를 투명하게** 보여줘 사용자로 하여금 기다릴지 취소할지 판단할 수 있게 합니다. **빈 상태**는 사용자 교육의 기회이기도 합니다 [40] – 예를 들어 처음 프로젝트가 없을 때 "아직 프로젝트가 없습니다. 새 프로젝트를 만들어보세요!" 라고 안내하고 CTA 버튼을 제공해 바로 행동을 취하도록 합니다 [41] [42]. 이는 사용자가 빈 화면에서 혼란스럽지 않도록 하고, 다음 행동을 촉진합니다. **에러 메시지**는 사용자 관점에서 이해할 수 있는 언어로 제공하고 (기술용어 X), 복구 방법을 함께 제시합니다 [14] [43]. 또한 치명적 오류 외에는 업무 흐름을 완전히 막지 않도록 설계합니다. 예를 들어 한 파일 업로드 실패했다고 전체 프로세스를 중단하기보다, 해당 파일만 오류 표시하고 다른 파일 처리는 계속하도록 합니다.

Pattern:
- **로딩 상태:**
- **전역 로딩 인디케이터:** 페이지 전환이나 주요 데이터 로딩 시 상단 네비게이션 바로 아래에 가는 **진행 막대(bar)**를 표시합니다 (YouTube 스타일). 이는 짧은 로드에도 사용자에게 진행 중임을 보여줍니다.
- **스켈레톤 vs 스피너:** 리스트나 카드 UI가 로딩될 때는 그 레이아웃을 본뜬 스켈레톤을 사용합니다 (예: 아바타 원, 텍스트 바 몇 개). 반면, 독립된 컴포넌트 하나만 로드할 땐 가운데 스피너를 쓰거나, 버튼 내 로딩 등으로 국지적으로 보여줍니다. 예: "분석 실행" 버튼 클릭 후 처리 중이면 버튼 내용을 스피너로 대체하고 disabled.
- **장기 작업 진행률:** 앞서 업로드, 파싱 등 5분 내외 작업은 **progress bar + 퍼센트 + 남은 시간** 패턴을 사용합니다. 너무 길 경우 "백그라운드 처리" 옵션 (체크박스나 "Continue in background" 버튼)을 제공해, 누르면 UI상표시는 숨기고 대시보드나 알림 영역에서 계속 추적하게 합니다.
- **취소 & 재시도:** 진행중 UI에 항상 "취소" (X 아이콘) 버튼을 보여주어, 사용자가 원하면 멈출 수 있게 합니다. 취소 시 현재까지 완료된 부분 처리(가능하다면)하고, 작업 목록에 "사용자 취소"로 기록합니다. 에러가 난 경우, 해당 UI에 "재시도" 버튼을 함께 표기하여, 누르면 재시작하도록 합니다.

    • **빈 상태(empty state):**

- **대시보드/프로젝트 목록 빈 화면:** “아직 프로젝트가 없습니다. 새 프로젝트를 만들어 AI 문서 작성을 시작해보세요.” 라는 메시지와 **[+ 새 프로젝트]** 주요 액션 버튼을 배치합니다. 적절한 일러스트 아이콘(예: 빈 파일 철)도 넣어 시선을 끕니다 [40] .
- **검색 결과 없음:** “검색 결과가 없습니다”와 함께, 검색어를 다시 확인하거나 필터를 풀어보라는 안내, 또는 FAQ 링크를 줍니다. (재밌는 문구보다는 명확한 안내 권장 [44]).
- **코멘트 없음:** “아직 코멘트가 없습니다. 팀원과 논의를 시작해보세요.” 정도와 코멘트 추가 버튼.
- **템플릿 매핑 처음 상태:** “왼쪽에서 필드를 선택하고 오른쪽 템플릿에 드롭하여 매핑을 시작하세요.”라는 튜토리얼식 안내를 중앙에 띄웁니다.

- 빈 상태에는 주로 **행동 유도(Call-to-Action)** 버튼을 함께 제공하여, 사용자가 빈 화면에서 뭘 해야 할지 알도록 합니다 [41] [45] . 예: 보고서 리스트 비면 “샘플 데이터 불러오기” 버튼을 제공해 체험을 돕는 것도 방법입니다.

- **에러 상태:**

- **양식 유효성 오류:** 앞서 설명한 것처럼 필드 옆에 바로 메시지를 표시합니다. 또한 폼 제출 시 전체적인 에러 요약을 상단에 표시(스크린리더 focus)해 “입력 오류가 있습니다. 필드를 확인해주세요.” 안내합니다.
- **네트워크/서버 오류:** 화면 중앙에 이모지와 함께 “서버와 통신에 실패했습니다. 인터넷 연결을 확인하거나 나중에 다시 시도하세요.” 등의 메시지. 재시도 버튼을 제공하며, 그래도 안되면 지원 연락처 안내. 500 에러 등은 사과 메시지와 함께, “오류 보고” 버튼으로 바로 로그를 첨부한 티켓을 열 수 있게 (또는 지원 이메일 링크 mailto:).
- **권한 거부(403):** “접근 권한이 없습니다. 관리자에게 요청하세요.” 메시지와 홈으로 돌아가기 링크.
- **찾을 수 없음(404):** “페이지를 찾을 수 없습니다. URL을 확인하세요.” 등 + 홈으로 이동 버튼.

- **긴 프로세스 에러:** 예를 들어 문서 생성 실패시, 해당 작업 카드에 “오류 발생: 원인 (예: 템플릿 오류). [로그 보기] [재시도]” 표시. 로그 보기를 누르면 상세 오류 로그를 모달이나 새 창에 표시해서 개발팀 공유 가능하게 합니다.

- **알림 & 작업 기록:**

- 토스트는 간단히 결과를 알려주지만 금방 사라지므로, **Notification Center** 또는 **작업 상세 패널**을 마련해 사용자가 이전 알림과 작업 로그를 볼 수 있게 합니다. 예: 상단 네비에 종 모양 아이콘 -> 누르면 최근 알림 드롭다운. 또는 대시보드에 “최근 작업” 리스트.
- 중요 오류의 경우 토스트 외에 화면 상Persistent 배너로도 알려, 사용자가 놓치지 않게 합니다. 예: “업로드 실패 - 네트워크 오류 (자세히)”.
- 모든 자동화 작업(생성, 내보내기 등)의 결과는 **감사로그**에도 남지만, 일반 사용자는 UI에 드러나는 활동 피드가 필요하므로, UI상 적절한 위치에 피드/로그를 노출합니다.

Implementation: 로딩 상태 구현은 Tailwind 등의 utility로 `animate-spin` (for spinner) 또는 skeleton 스타일(css `@keyframes pulse`)을 적용해 만들고, React state로 데이터 로딩 여부를 제어합니다. Skeleton 컴포넌트는 각 페이지별로 제작해 placeholder를 나타내고, 데이터 도착 시 fade-out 또는 instant replace. 장기 작업은 progress state를 관리하고, context나 Redux로 다른 컴포넌트에도 전달 가능하게 해, 예컨대 header나 favicon에 progress 표시도 고려 가능합니다. 빈 상태는 컴포넌트가 list length=0인 경우 조건부 렌더링으로 별도 EmptyState component를 보여주는 식. 에러는 try-catch나 API error handling에서 에러 메시지를 상태로 세팅해 Error boundary 컴포넌트로 보여줍니다. React ErrorBoundary를 만들어, 치명적 오류 시 UI를 대체 “Oops” 화면으로 전환시키고 Sentry 로깅 등을 할 수 있습니다.

오류 메시지 텍스트는 국제화되어야 하므로 리소스에 키로 관리합니다. 또한 개발/QA 시 의도적으로 에러 상황(오프라인 모드 등)을 테스트해 UI를 튜닝합니다.

Notification Center는 간단히는 상태로 알림 배열을 관리하고, 벨 아이콘 클릭 시 dropdown으로 mapping. 각 알림 아이템은 타입(success/warning/error 등)과 메시지, timestamp, 링크 (로그 등) 를 포함. 더 나아가 브라우저 Notification API를 이용해 권한 허용 시 OS 알림도 보낼 수 있습니다 (긴 작업 완료 시).

마지막으로, **상태 별 UI 일관성**을 위해 디자인 가이드에 각 상황별 예시를 명시하고, Storybook에서 "Loading Empty Error States" 스토리를 만들어 컴포넌트별 시나리오 (예: Table with no data, Image failed to load, etc.)를 모두 검증합니다 [19] [41] . 이렇게 준비된 패턴은 향후 새로운 화면에도 적용되어, 전체 제품의 UX를 균일하게 유지할 것입니다.

Sources: 각주에서 【】로 표시된 숫자는 참고한 자료를 나타냅니다. 이 명세에서는 Material Design 가이드, Nielsen Norman Group의 UX 아티클, Tailwind CSS 및 다양한 기술 문서에서 모범 사례와 수치를 인용하여 근거를 명시하였습니다. 이러한 **Problem → Principle → Pattern → Implementation** 체계의 심층 리서치 결과는 추후 설계 변경이나 구현 시에도 일관된 판단 기준을 제공하며, XML 등으로 구조화하여 **UI/UX 코드 생성 자동화**에도 활용할 수 있을 것입니다.

---

1 How to Build a Role-Based Access Control Layer
https://www.osohq.com/learn/rbac-role-based-access-control

2 3 Menu-Design Checklist: 17 UX Guidelines - NN/G
https://www.nngroup.com/articles/menu-design/

4 5 6 7 9 How to Build a Design Token System for Tailwind That Scales Forever | by Hex Shift | Medium
https://hexshift.medium.com/how-to-build-a-design-token-system-for-tailwind-that-scales-forever-84c4c0873e6d

8 39 Website Color Palettes for Accessibility
https://www.concretecms.com/about/blog/web-design/inclusive-website-color-palettes-for-accessibility

10 11 Theme variables - Core concepts - Tailwind CSS
https://tailwindcss.com/docs/theme

12 16 20 36 37 38 WCAG 2.2 Explained: New Accessibility Success Criteria and Requirements
https://www.audioeye.com/post/wcag-22/

13 14 15 22 43 44 Error-Message Guidelines - NN/G
https://www.nngroup.com/articles/error-message-guidelines/

17 Data Table From Scratch. Part 3: Virtualization - DEV Community
https://dev.to/morewings/lets-create-data-table-part-3-virtualization-5778

18 19 41 42 45 Designing Empty States in Complex Applications: 3 Guidelines - NN/G
https://www.nngroup.com/articles/empty-state-interface-design/

21 Making form error messages accessible - HSBC Accessibility Hub
https://www.inclusion.hsbc.com/en/articles/common-mistakes/error-messages-in-forms

23 24 Skeleton Screens vs. Loading Screens -- An UX Battle
https://blog.openreplay.com/skeleton-screens-vs-loading-screens--a-ux-battle/

25 UX Design Patterns for Loading - Pencil & Paper
https://www.pencilandpaper.io/articles/ux-pattern-analysis-loading-feedback

26 Design Elements of Search - Results
https://findwise.com/blog/design-elements-search-results/

27 28 29 30 Building Trust in LLM Answers: Highlighting Source Texts in PDFs | by SO Development | Medium
https://sodevelopment.medium.com/building-trust-in-llm-answers-highlighting-source-texts-in-pdfs-0a9d26417353

31 32 Working with placeholders — python-pptx 1.0.0 documentation
https://python-pptx.readthedocs.io/en/latest/user/placeholders-using.html

33 Audit logs | UI Bakery Docs
https://docs.uibakery.io/concepts/workspace-management/audit-logs

34 35 Audit Logs: Introducing the New Design Experience (UI)
https://help.gohighlevel.com/support/solutions/articles/155000006667-audit-logs-introducing-the-new-design-experience

40 do empty state design best practices actually matter for conversion ...
https://www.reddit.com/r/webdev/comments/1pzjrbr/do_empty_state_design_best_practices_actually/