[기획서] 인사팀 관리 시스템 (HRM CMS) MVP(2)

인사팀 CMS(HRM) 구축 기획서 (Draft)

이 기획서는 나중에 **커서(Cursor)나 제미나이(Gemini)**에게 "이 기획서대로 코드를 짜줘"라고 요청하기 최적화된 구조로 작성되었습니다.

1. 프로젝트 개요

- 목적: 직원 정보 관리, 근태 관리, 공지사항 운영을 위한 인사팀 전용 관리 도구 구축
- 핵심 원칙: * P0 우선 구현: 핵심 운영에 필요한 필수 기능을 우선 개발
- 데이터 보안: 모든 통신은 Base44 기반 로직을 적용하여 무결성 유지
- 사용성: 데이터 테이블 중심의 직관적인 UI 제공

2. 인포메이션 아키텍처 (IA) & 상세 기능

1Depth (메뉴)	2Depth (화면)	우선순위(p1)	핵심기능 (P0/P1)	상세 설계
인증	로그인	P1	주요 통계(총원, 휴가자), 최근 활동	ID, PW, 자동로그인 여부
현황	대시보드	P0	주요 통계(총원, 휴가자), 최근 활동	통계 차트, 로그 리스트
직원 관리	직원 목록	P0	데이터 목록 조회, 페이지네이션, 검색/필터(P1)	사번, 이름, 부서, 직급, 상태

	직원 상세		데이터 상세 조회, 개별/일괄 삭제(P1)	상세 인적사항, 이력
	직원 등록/수정		데이터 등록 및 기존 데이터 편집	입력 폼 (이름, 사번 등)
근태 관리	근태 목록	P0	휴가/근태 데이터 조회 및 상태 변경	신청인, 기간, 사유, 상태
시스템	설정 페이지	P1	권한 관리 (Admin/User), 계정 관리	계정명, 권한 등급

ㄱ) 기획서 2번 섹션 권한 및 접근 제어 로직

```
/**
* [권한 규칙] RoleBasedView
* 기획서 P1: Admin/User 권한 분리 적용
* - Admin: 모든 기능(조회, 등록, 수정, 삭제, 설정) 가능
* - User: 조회 및 등록만 가능 (삭제 및 설정 접근 불가)
*/

import React from 'react';

// 권한에 따라 컴포넌트를 노출하는 래퍼 컴포넌트
export const RoleBasedView = ({ allowedRoles, userRole, children }) => {
// 현재 유저의 권한이 허용된 권한 목록에 있는지 확인
if (!allowedRoles.includes(userRole)) {
```

```
return null; // 권한이 없으면 아무것도 보여주지 않음

}

return <>{children}</>;

};



// 사용 예시 (기획서 상세 설계 반영)

// <RoleBasedView allowedRoles={['Admin']} userRole={currentUser.role}>

// <button className="delete-btn">일괄 삭제 (P1)</button>

// </RoleBasedView>
```

ㄴ) IA와 React 폴더 구조 매칭 방법

```
src/

├── components/ # 공통 UI (Button, Modal, Layout 등)

├── utils/ # 공통 로직 (base44Crypto.js)

├── pages/ # IA의 1Depth/2Depth 화면 단위

│   ├── auth/ # 1Depth: 인증

│   │   └── LoginPage.jsx

│   ├── dashboard/ # 1Depth: 현황

│   │   └── DashboardPage.jsx

│   ├── staff/ # 1Depth: 직원 관리

│   │   ├── StaffListPage.jsx # 2Depth: 직원 목록

│   │   ├── StaffDetailPage.jsx # 2Depth: 직원 상세

│   │   └── StaffFormPage.jsx # 2Depth: 등록/수정 폼

│   └── attendance/ # 1Depth: 근태 관리
```

```
|  └── AttendancePage.jsx
└── App.js # 전체 라우팅 (유저 플로우 설계 반영)
```

3. 유저 플로우

```
graph TD
A[로그인 화면 P0] --> B{인증 완료}
B -->|Yes| C[대시보드 P1]
C --> D[직원 목록 페이지 P0]
D -->|키워드 검색/필터 P1| D
D -->|등록 버튼| E[데이터 등록 폼 P0]
D -->|행 클릭| F[데이터 상세 페이지 P0]
D -->|체크 후 일괄 삭제 P1| D
F -->|수정 버튼| G[데이터 수정 폼 P0]
F -->|개별 삭제 P1| D
E -->|Base44 저장 요청| D
G -->|Base44 수정 요청| F
```

4. 상세 화면 구성 (UI Structure)

① 로그인 페이지 (P0)

- UI: 중앙 정렬된 로그인 카드, 이메일/비밀번호 입력창.

- 기능: 유효성 검사, 로그인 실패 시 에러 메시지 표시.

② 목록 페이지 (P0/P1)

- 상단 (P1): 키워드 검색창(이름/사번), 부서/상태별 필터 드롭다운.

- 중앙 (P0): 데이터 테이블 (사번, 이름, 부서, 직급, 상태 열 포함).

- 액션 (P1): 테이블 좌측 체크박스를 통한 '일괄 삭제' 기능.

- 하단 (P0): 10개 단위 페이지네이션.

③ 상세 페이지 (P0)

- 내용: 직원의 모든 필드 정보를 읽기 전용으로 표시.

- 액션: 수정 페이지 이동 버튼, 삭제 버튼(P1).

④ 등록/수정 폼 (P0)

- UI: 각 데이터 필드별 입력창 (Text, Select, DatePicker).

- 기능: 저장 시 Base44 로직을 호출하여 서버와 통신.

ㄱ) 기획서 4번 섹션 React 전체 뼈대

*** 권한제한 규칙 기술 적용**

: AuthContext를 생성하여 로그인한 유저의 세션 정보(이름, 사번) 및 권한(Role: Admin/User)을 전역 상태로 관리한다.

메뉴 제어 (GNB): 유저의 Role 값에 따라 GNB(Global Navigation Bar) 메뉴 노출을 동적으로 제어한다. (예: 시스템 설정 메뉴는 Admin에게만 노출)

기능 제어: 상세 페이지 및 목록 페이지 내의 '삭제'나 '수정' 버튼은 Admin 권한이 있을 때만 활성화(Active)한다.

기획 의도: UX 원칙인 "3번의 클릭 내 도달"을 위해 유저 권한에 불필요한 메뉴를 사전에 차단하여 최적화된 경로를 제공한다.

```
/**
* [구조 규칙] App.js (전체 라우팅 및 접근 제어)
* IA 위계 구조를 따르며, 유저 플로우에 정의된 경로로만 이동 가능함[cite: 10, 13, 14].
*/
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';

function App() {
return (
<Router>
<Routes>
{/* 인증: 로그인 페이지 P0 [cite: 11, 16] */}
<Route path="/login" element={<LoginPage />} />

{/* 보호된 경로: 인증 완료된 사용자만 진입 가능 [cite: 14] */}
<Route path="/" element={<Layout />}>
<Route index element={<Navigate to="/dashboard" replace />} />
{/* 현황: 대시보드 P1 [cite: 11] */}
<Route path="dashboard" element={<DashboardPage />} />
{/* 직원 관리: 목록 P0, 상세 P0, 등록/수정 P0 [cite: 11] */}
<Route path="staff" element={<StaffListPage />} />
<Route path="staff/:id" element={<StaffDetailPage />} />
<Route path="staff/form" element={<StaffFormPage />} />
{/* 근태 관리: 목록 P0 [cite: 12] */}
<Route path="attendance" element={<AttendancePage />} />
```

```
{/* 시스템 설정: 관리자 전용 P1 [cite: 12, 33] */}
<Route path="settings" element={<SettingsPage />} />
</Route>
</Routes>
</Router>
);
}
```

ㄴ) 기획서 4번 섹션 상세화면 REACT 2-1

보안 원칙: 해당 페이지는 5번 기술 및 보안 요구사항의 API 통신 규격을 엄격히 준수한다.

데이터 해독: 모든 응답 데이터는 화면 렌더링 직전 반드시 hrmCrypto.decode를 호출하여 평문으로 복원해야 한다.

상태 관리: useState를 통해 해독된 직원 데이터를 관리하며, 데이터 무결성을 유지한다.

자동 로딩: useEffect를 활용하여 페이지 접속 시 자동으로 서버 데이터를 호출하고 보안 로딩 프로세스를 시작한다.

* 공통 에러 핸들링 및 피드백 규칙

- "데이터 로드 실패 시 **5-2 규칙**에 따라 에러 페이지로 이동함."

```
/**
* [화면 규칙] StaffListPage.js (직원 목록 및 검색)
* 1. 데이터 수신: API로부터 받은 Base44 데이터를 hrmCrypto.decode로 풀어서 출력함.
* 2. 화면 구성: 상단 검색바(P1), 중앙 데이터 테이블(P0), 하단 페이지네이션(P0).
* 3. UX 원칙: 행 클릭 시 상세 페이지(/staff/:id)로 3번의 클릭 안에 이동.
*/
```

```
import React, { useState, useEffect } from 'react';
import { hrmCrypto } from '../utils/base44Crypto';

const StaffListPage = () => {
const [staffList, setStaffList] = useState([]);
const [searchTerm, setSearchTerm] = useState("");

// 데이터 로드 시 보안 규칙 적용
const fetchStaffData = async () => {
const response = await fetch('/api/staff');
const encodedData = await response.json();
// 기획서 보안 원칙: 수신 데이터는 반드시 디코딩 후 상태에 저장
const decodedData = hrmCrypto.decode(encodedData);
setStaffList(decodedData);
};

return (
<div className="staff-list-container">
<h2>직원 목록 관리</h2>
{/* 검색 필터 영역 (P1) */}
<div className="search-bar">
<input
type="text"
```

```
placeholder="사번 또는 이름으로 검색"

onChange={(e) => setSearchTerm(e.target.value)}

/>

<button>조회</button>

</div>


{/* 데이터 테이블 (P0) */}

<table className="staff-table">

<thead>

<tr>

<th>사번</th>

<th>성명</th>

<th>부서</th>

<th>직급</th>

<th>상태</th>

<th>관리</th>

</tr>

</thead>

<tbody>

{staffList.map((staff) => (

<tr key={staff.id}>

<td>{staff.staff_id}</td>

<td>{staff.name}</td>

<td>{staff.department}</td>
```

```
<td>{staff.position}</td>

<td>{staff.status}</td>

<td><button>상세보기</button></td>

</tr>

))}

</tbody>

</table>

{/* 페이지네이션 (P0): 10개 단위 노출 규칙 */}

<div className="pagination">

<button>이전</button>

<span>1 / 5</span>

<button>다음</button>

</div>

</div>

);

};
```

ㄹ) 등록/수정 화면 구현 명세 REACT

*** 지침:** API 요청(Request)을 보내기 직전에 모든 데이터를 hrmCrypto.encode로 변환할 것.

* **준수 사항**: 해당 페이지는 5번 기술 및 보안 요구사항의 API 통신 규격을 준수하며, 모든 요청 데이터는 전송 직전 hrmCrypto.encode로 변환되어야 함.

*** * 공통 에러 핸들링 및 피드백 규칙**

- 저장 실패 시 5-2 규칙에 따라 Toast 메시지를 노출함.

```
/**
```

```
* [화면 규칙] StaffFormPage.js (직원 등록 및 수정)
* 1. 데이터 검증: 필수 항목(사번, 이름, 부서) 입력 여부 확인.
* 2. 보안 전송: 전송 전 모든 데이터를 hrmCrypto.encode로 인코딩함.
* 3. UX 원칙: 저장 완료 후 '직원 목록(/staff)'으로 자동 이동하여 흐름 유지.
*/

import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { hrmCrypto } from '../utils/base44Crypto';

const StaffFormPage = ({ mode = 'create', initialData = {} }) => {
const navigate = useNavigate();
// DB 스키마와 1:1 매칭되는 상태 관리
const [formData, setFormData] = useState({
staff_id: initialData.staff_id || '',
name: initialData.name || '',
department: initialData.department || '',
position: initialData.position || '',
status: initialData.status || '재직'
});

const handleSubmit = async (e) => {
e.preventDefault();
```

```
// 1. 보안 규칙 적용: 전송 데이터를 Base44로 인코딩

const encodedPayload = hrmCrypto.encode(formData);



// 2. API 전송 (예시)

const response = await fetch('/api/staff/save', {

method: 'POST',

body: JSON.stringify({ data: encodedPayload }),

headers: { 'Content-Type': 'application/json' }

});



if (response.ok) {

alert("성공적으로 저장되었습니다.");

// 3. 유저 플로우 규칙: 목록으로 복귀

navigate('/staff');

}

};



return (

<div className="staff-form-container">

<h2>{mode === 'create' ? '신규 직원 등록' : '직원 정보 수정'}</h2>

<form onSubmit={handleSubmit}>

<div className="form-group">

<label>사번</label>

<input
```

```
value={formData.staff_id}

onChange={(e) => setFormData({...formData, staff_id: e.target.value})}

required

/>

</div>

<div className="form-group">

<label>성명</label>

<input

value={formData.name}

onChange={(e) => setFormData({...formData, name: e.target.value})}

required

/>

</div>

{/* 부서, 직급 등 나머지 입력 필드 동일 구조 */}

<div className="form-actions">

<button type="button" onClick={() => navigate(-1)}>취소</button>

<button type="submit" className="save-btn">저장</button>

</div>

</form>

</div>

);

};
```

5. 기술 및 보안 요구사항

- **데이터 전송:** 프론트엔드와 백엔드 간 모든 JSON 데이터는 Base44 규칙에 따라

인코딩/디코딩 처리.

- **권한 관리 (P1)**: * Admin: 모든 기능(삭제, 설정 포함) 접근 가능.

- **User:** 목록 조회, 상세 조회, 등록만 가능 (삭제 및 설정 접근 제한).

***지침**:모든 인사 정보 데이터는 네트워크 전송 시 평문 노출을 방지하기 위해 Base44 기반의 암호화를 수행한다.

모든 API 요청 body는 encodeData 함수를 거쳐야 함

ㄱ) 보안 유틸리티 코드 REACT(BASS44)

* **5-1)** 모든 API 데이터는 hrmCrypto 모듈을 통해 Base44 인코딩/디코딩을 거친다.

* **5-2) *공통 에러 핸들링 및 피드백 규칙**

- API 통신 실패: 서버 연결 불가, 404/500 에러 등 API 호출 실패 시, 즉시 Toast 메시지를 통해 유저에게 상황을 알린다. (예: "데이터 처리 중 오류가 발생했습니다. 잠시 후 다시 시도해 주세요.")

- **보안 해독(Decoding) 실패:** 데이터 수신 후 hrmCrypto.decode 과정에서 에러가 발생할 경우, 깨진 데이터를 화면에 보여주는 대신 에러 전용 페이지로 리다이렉트하거나 목록 새로고침을 유도한다.

- **입력값 검증:** 등록/수정 시 필수 항목이 누락된 경우 서버로 데이터를 보내기 전 브라우저 단에서 1차 차단하고 알림을 띄운다.

- **기획 의도:** 시스템 장애나 보안 오류 발생 시 사용자가 방치되지 않도록 즉각적이고 친절한 피드백을 제공하여 서비스 신뢰도를 높인다.

5-3) 시스템 환경 설정 및 환경 변수

- *중앙 관리:* 서버 주소, 페이지네이션 개수 등 변경 가능성이 있는 설정값은 src/config.js 파일에서 통합 관리한다.

- 주요 설정 항목:

API_BASE_URL: 백엔드 서버의 기본 주소.

ITEMS_PER_PAGE: 목록 페이지당 노출 데이터 개수 (기본값: 10).

CRYPTO_KEY: Base44 인코딩 시 사용할 보조 키 (필요시).

- **기획 의도:** 유지보수 효율성을 극대화하여, 서버 이전이나 정책 변경(예: 10개 -> 20개 노출) 시 코드 전체를 수정하지 않고 설정 파일만 변경하여 대응한다.

```
/**
* [보안 규칙] base44Crypto.js
* 모든 통신 데이터(JSON)는 전송 직전 인코딩, 수신 직후 디코딩 과정을 거쳐야 함[cite: 31].
*/
export const hrmCrypto = {
// 데이터를 서버로 보낼 때 (React -> API) [cite: 29]
encode: (data) => {
try {
const jsonStr = JSON.stringify(data);
// UTF-8 보정 후 Base44 인코딩 [cite: 31, 32]
return btoa(unescape(encodeURIComponent(jsonStr)));
} catch (e) {
console.error("Encoding Error", e);
return null;
}
},

// 서버에서 데이터를 받을 때 (API -> React) [cite: 31]
```

```
decode: (encodedStr) => {
try {
// Base44 디코딩 후 객체 복원 [cite: 32]
const decodedStr = decodeURIComponent(escape(atob(encodedStr)));
return JSON.parse(decodedStr);
} catch (e) {
console.error("Decoding Error", e);
return null;
}
}
};
```

6. 평가 기준 (Acceptance Criteria)

[ ] P0: 로그인 후 직원 목록이 테이블에 정상적으로 출력되는가?

[ ] P0: 등록 폼을 통해 추가한 데이터가 상세 페이지에 반영되는가?

[ ] P1: 이름 검색 시 해당 데이터만 테이블에 필터링되는가?

[ ] P1: 관리자 권한이 없는 계정은 '삭제' 버튼이 비활성화되는가?

1. 기획서 AI에게 파일업로드

2. 기획서를 참고하여 머메이드 구조 만들어줘

- 로그인 → 대시보드 → 직원 목록 → 등록/수정"으로 이어지는 전체 흐름도

- 이 내용을 나중에 커서(Cursor)에 복사해 넣으면 바로 코딩 설계도로 쓰실 수 있음

3. 유저플로우 작성(머메이드(Mermaid) 문법을 사용하여 텍스트로 논리를 짜고 이를 차트로 변환)

1) 유저 플로우 다이어그램 (User Flow)

```
graph TD

%% 인증 단계

Start((시작)) --> Login[로그인 화면 P0]

Login --> Auth{인증 완료?}

Auth -- No --> Login

Auth -- Yes --> Dash[대시보드 P1]


%% 메인 메뉴 이동

Dash --> List[직원 목록 페이지 P0]

Dash --> Attendance[근태 목록 P0]

Dash --> Setting[설정 페이지 P1]


%% 직원 관리 상세 로직

List --> Search[키워드 검색/필터 P1]

List --> RegForm[데이터 등록 폼 P0]

List --> Detail[데이터 상세 페이지 P0]

List --> BulkDel[체크 후 일괄 삭제 P1]


%% 수정 및 저장 (Base44 적용)

RegForm -- "Base44 저장 요청" --> List

Detail --> EditForm[데이터 수정 폼 P0]

Detail --> SingleDel[개별 삭제 P1]

SingleDel --> List

EditForm -- "Base44 수정 요청" --> Detail
```

```
%% 스타일 정의
style Start fill:#f9f,stroke:#333
style Login fill:#fff4dd,stroke:#d4a017
style Base44 fill:#e1f5fe,stroke:#01579b
```

ㄱ) 유저 플로우 섹션용 React

```
/**
* [유저 플로우 규칙] App.js & ProtectedRoute
* 1. 인증 가드: 로그인하지 않은 유저는 모든 경로에서 /login으로 리다이렉트됨.
* 2. 흐름 제어: useNavigate를 사용하여 유저 플로우에 정의된 순서대로 이동.
*/
import { BrowserRouter, Routes, Route, Navigate, useNavigate } from 'react-router-dom';

// 인증 확인용 가드 컴포넌트
const ProtectedRoute = ({ children }) => {
const isAuthenticated = localStorage.getItem('token'); // 로그인 여부 확인
return isAuthenticated ? children : <Navigate to="/login" replace />;
};

function AppRouter() {
return (
<BrowserRouter>
<Routes>
```

```
{/* 유저 플로우 A: 로그인 */}
<Route path="/login" element={<LoginPage />} />

{/* 유저 플로우 B 이후 (인증 필요) */}
<Route path="/" element={<ProtectedRoute><Layout /></ProtectedRoute>}>
<Route path="dashboard" element={<DashboardPage />} /> {/* C: 대시보드 */}
<Route path="staff" element={<StaffListPage />} /> {/* D: 직원 목록 */}
<Route path="staff/form" element={<StaffFormPage />} /> {/* E: 등록 폼 */}
<Route path="staff/:id" element={<StaffDetailPage />} /> {/* F: 상세 페이지 */}
</Route>
</Routes>
</BrowserRouter>
);
}
```

2) 인포메이션 아키텍처 위계 구조 (IA Tree)

```
mindmap
root((인사팀 CMS))
인증 P0
로그인
ID/PW 입력
자동로그인
현황 P1
대시보드
```

통계 차트

최근 활동 로그

직원 관리 P0

직원 목록

검색 및 필터 P1

데이터 테이블

페이지네이션

직원 상세

상세 인적사항

개별 및 일괄 삭제 P1

등록 및 수정

입력 폼

Base44 통신

근태 관리 P0

근태 목록

조회 및 상태 변경

시스템 P1

설정 페이지

권한 관리 Admin/User

계정 관리

3) 기획서 핵심 요약 및 적용 포인트

이 머메이드 구조를 기반으로 개발을 진행하실 때 참고할 핵심 포인트입니다:

- **보안 전송:** 모든 데이터 등록 및 수정 시 Base44 기반 로직을 호출하여 서버와 통신해야

합니다.

- **권한 분리:** Admin은 모든 권한을 가지지만, User는 목록/상세 조회 및 등록만 가능하며 삭제와 설정은 제한됩니다.

- **목록 구성:** 테이블은 10개 단위 페이지네이션을 기본으로 하며, 상단에 검색 및 필터 기능이 포함됩니다.

*참고: React → API → DB 순서가 갖춰야만 실제로 구현함

4) 데이터 베이스 만들기

- 사용자가 입력한 정보를 저장하는 창고임

- 저장될 데이터의 규격(사번은 숫자, 이름은 문자 등)을 먼저 확정 짓는 것

(1) 인사팀 CMS 데이터베이스 설계 (DB Schema)

- 기획서의 **P0(필수)**와 P1(권장) 데이터를 모두 수용할 수 있도록 설계했습니다.

ㄱ) 직원 정보 테이블 (Staff)

- 가장 기본이 되는 창고로, 직원 한 명 한 명의 명부를 담습니다

컬럼명 (Column)	데이터 타입 (Type)	필수 여부	설명
id	INT (PK)	필수	고유 식별 번호 (자동 생성)
staff_id	VARCHAR(20)	필수	사번 (중복 불가)
name	VARCHAR(50)	필수	이름
department	VARCHAR(50)	필수	부서
position	VARCHAR(30)	필수	직급
status	ENUM	필수	상태 (재직, 휴직, 퇴사)

email	VARCHAR(100)	필수	ID로 사용될 이메일
password	VARCHAR(255)	필수	암호화된 비밀번호
join_date	DATE	필수	입사일

ㄴ) 근태/휴가 관리 테이블 (Attendance)

- 직원별 근태 기록과 휴가 신청 내용을 담는 창고입니다.

컬럼명 (Column)	데이터 타입 (Type)	필수 여부	설명
att_id	INT (PK)	필수	고유 번호
staff_id	VARCHAR(20)	필수	직원 테이블과의 연결 고리 (FK)
type	ENUM	필수	구분 (출근, 퇴근, 휴가)
start_date	DATETIME	필수	시작 일시 (기간)
end_date	DATETIME	필수	종료 일시 (기간)
reason	TEXT	선택	사유
status	ENUM	필수	결재 상태 (대기, 승인, 반려)

🔗 참고

공간이 모자랄 때를 대비한 '유연한 설계'

- **정규화:** 직원 정보와 근태 정보를 분리하여, 직원이 늘어나거나 근태 기록이 수만 건 쌓여도 시스템이 느려지지 않게 설계했습니다.

- **Base44 및 암호화 공간:** 비밀번호나 민감 정보(password)는 Base44 또는 더 강력한 - - 암호화 데이터가 담길 수 있도록 글자 수 제한을 넉넉하게(255자 이상) 설정했습니다.

- **확장성:** 나중에 '급여 관리'나 '심리 리포트' 기능이 추가되더라도 기존 staff_id를 활용해 새로운 창고(테이블)만 옆에 더 지으면 되도록 설계했습니다

. 와이어프레임 (Wireframe) 시각화

상담내용은 : AES-256 같은 더 강력한 암호화 방식을 추가