IITP DABT Admin

프로젝트 아키텍처

문서 버전: 1.0.0
작성일: 2025-11-07

(주)스위트케이

문서 History

버전	일자	작성자	변경 내용	비고
1.0.0	2025-11-07	(주)스위트케이	최초 작성	

목차

1. 프로젝트 개요

1.1 프로젝트 소개

1.1.1 프로젝트 배경 및 목적

IITP DABT Admin은 Open API 센터 및 관리자 시스템입니다.

- OpenAPI 사용 인증키 관리 및 게정 관리
- 콘텐츠 관리 (FAQ, Q&A, 공지사항)

1.1.2 시스템 개요

본 시스템은 다음과 같은 구성으로 이루어져 있습니다:

```
Client (Web Browser)
        ↓
Frontend (React SPA)
        ↓
Backend (Express API)
        ↓
Database (PostgreSQL)
```

1.2 시스템 범위

1.2.1 기능 범위

인증 및 계정 관리:

- 사용자 회원가입 및 로그인
- 관리자 로그인 (역할별 차등 권한)
- 프로필 관리

- 비밀번호 변경

OpenAPI 서비스:

- API 인증 키 발급
- 키 목록 조회 및 관리
- 키 활성화/비활성화
- 키 유효기간 관리 및 연장

콘텐츠 관리:

- FAQ 관리 (생성, 수정, 삭제, 조회)
- Q&A 관리 (질문 접수, 답변 작성, 상태 관리)
- 공지사항 관리 (생성, 수정, 삭제, 게시 기간 설정)

시스템 관리:

- 관리자(운영자) 계정 관리
- 공통 코드 관리

1.2.2 기술 및 인프라 범위

프로젝트 구조:

- Monorepo 구조 (Common, Backend, Frontend)
- npm Workspaces 기반 패키지 관리

Backend:

- Node.js 22.x 기반 Express.js API 서버
- TypeScript 5.x
- PostgreSQL 12.x 데이터베이스
- Sequelize 6.x ORM

Frontend:

- React 18.x 기반 SPA
- TypeScript 5.x
- Vite 5.x 빌드 도구
- Material-UI 5.x UI 라이브러리

인프라:

- Nginx 웹서버 (정적 파일 서빙 및 API 프록시)
- PM2 프로세스 관리
- Winston 로깅 시스템

1.3 주요 기능 요약

1.3.1 사용자 기능 (User)

일반 사용자 사용 가능 기능:

- 회원가입 및 로그인
- 본인 프로필 조회/수정
- OpenAPI 키 발급 및 관리
- FAQ 조회
- Q&A 작성 및 조회
- 공지사항 조회

1.3.2 관리자 기능 (Admin)

관리자 사용 가능 기능:

- 관리자 대시보드
- 사용자 관리
- 콘텐츠 관리 (FAQ, Q&A, 공지사항)
- OpenAPI 키 관리 (모든 사용자)
- 운영자 계정 관리 (S-ADMIN 전용)
- 공통 코드 관리 (S-ADMIN 전용)

1.3.3 시스템 관리 기능

시스템 운영 및 모니터링:

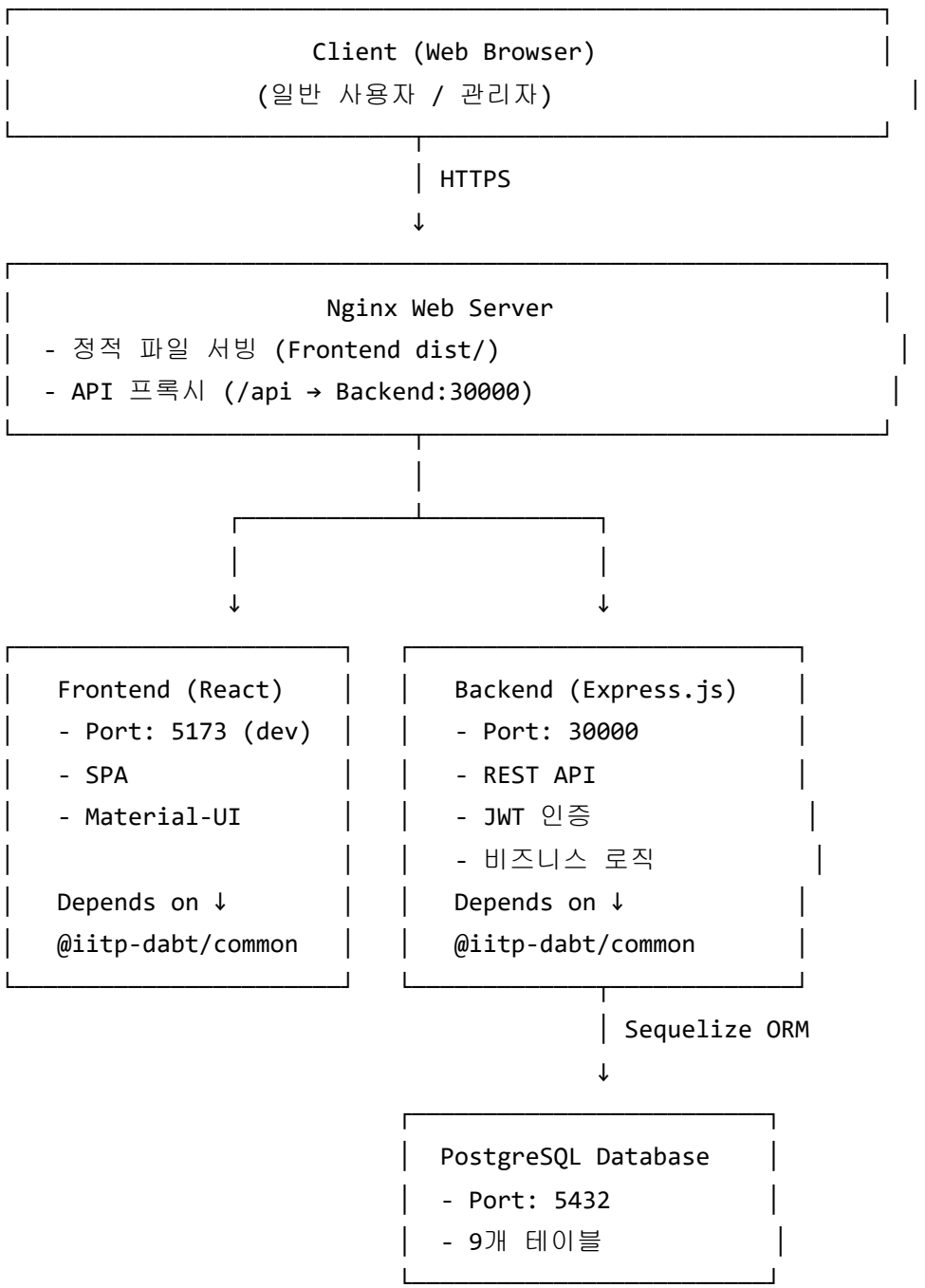
- 사용자 접근 로그 기록
- 데이터 변경 이력 추적
- 에러 로그 관리
- 시스템 상태 모니터링

1.4 참고 문서

- **IITP DABT Admin Backend 상세 설계서** : Backend 상세 설계서
- **IITP DABT Admin Frontend 상세 설계서** : Frontend 상세 설계서
- API 규격서 : API 스펙 상세
- 서버 배포 및 설치 가이드 : 서버 배포/설치/실행 가이드

2. 시스템 아키텍처

2.1 전체 시스템 구성도



2.2 아키텍처 개요

2.2.1 Monorepo 구조

Monorepo 구조로 Common/BE/FE 패키지들을 하나의 저장소에서 관리합니다.

구조:

```
IITP-DABT-Admin/
├── packages/
│   └── common/          # 공통 패키지
├── be/                  # Backend
├── fe/                  # Frontend
├── script/              # 빌드/배포 스크립트
└── package.json         # 루트 package.json (Workspace 설정)
```

2.2.2 3-Tier 아키텍처

Frontend (Presentation Layer):

- 사용자 인터페이스
- 사용자 입력 처리
- 상태 관리
- 라우팅 및 화면 접근 권한 제어

Backend (Application Layer):

- REST API 제공
- 비즈니스 로직 처리
- 인증/인가
- 데이터 검증

Database (Data Layer):

- 데이터 영속성
- 트랜잭션 관리
- 데이터 무결성

2.3 Common 패키지*

2.3.1 Common 패키지 기능

BE/FE 공통 처리를 위해 Common 패키지를 사용합니다.

- API 규격 정의
- Single Source of Truth
- 검증 로직 일관성 보장
- TypeScript 타입 자동 공유
- 에러 코드 및 권한 코드 통일
- 유지보수 용이

예시:

```
// packages/common/src/validation.ts
export function isValidEmail(email: string): boolean {
  return /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/.test(email);
}

// Backend에서 사용
import { isValidEmail } from '@iitp-dabt/common';

// Frontend에서 사용
import { isValidEmail } from '@iitp-dabt/common';
```

2.3.2 Common 패키지의 기능

2.3.2.1. 검증 로직 통일

제공 함수(예시):

```
isValidEmail(email: string): boolean
isValidPassword(password: string): boolean
validatePassword(password: string): { isValid: boolean; errorMessage?: string }
getPasswordStrength(password: string): 'weak' | 'medium' | 'strong'
isValidName(name: string): boolean
isValidAffiliation(affiliation: string): boolean
```

처리:

- **Backend**: API 요청 검증 (Controller 단계)
- **Frontend**: 실시간 입력 검증 (Form 컴포넌트)

목적:

- 사용자가 Frontend에서 입력 → Frontend 검증 통과 → Backend 전송 → Backend 검증 통과
- 동일한 검증 로직으로 **일관성 보장**

2.3.2.2. 타입 정의 공유 (types/)

API 타입 정의:

```
// 요청/응답 인터페이스
interface LoginRequest {
  email: string;
  password: string;
}

interface LoginResponse {
  accessToken: string;
  refreshToken: string;
  user: UserProfile;
}

interface ApiResponse<T> {
  result: 'ok' | 'error';
  data?: T;
  message?: string;
  errorCode?: number;
}
```

처리:

- **Backend**: Controller에서 요청/응답 타입 정의
- **Frontend**: API 함수에서 요청/응답 타입 정의

목적:

- TypeScript 컴파일 타임에 타입 불일치 자동 감지
- API 스펙 변경 시 BE/FE 동시 반영

2.3.2.3. 에러 코드 통일 (errorCodes.ts)

에러 코드 체계:

```
enum ErrorCode {
  // 기본 에러 (11xxx)
  UNKNOWN_ERROR = 11000,
  VALIDATION_ERROR = 11001,
  DATABASE_ERROR = 11002,

  // 인증 관련 (14xxx)
  UNAUTHORIZED = 14000,
  TOKEN_EXPIRED = 14003,
  FORBIDDEN = 14008,

  // 사용자 관련 (16xxx)
  USER_NOT_FOUND = 16000,

  // FAQ 관련 (22xxx)
  FAQ_NOT_FOUND = 22000,
  FAQ_CREATE_FAILED = 22001,

  // ...
}
```

에러 메시지 매핑:

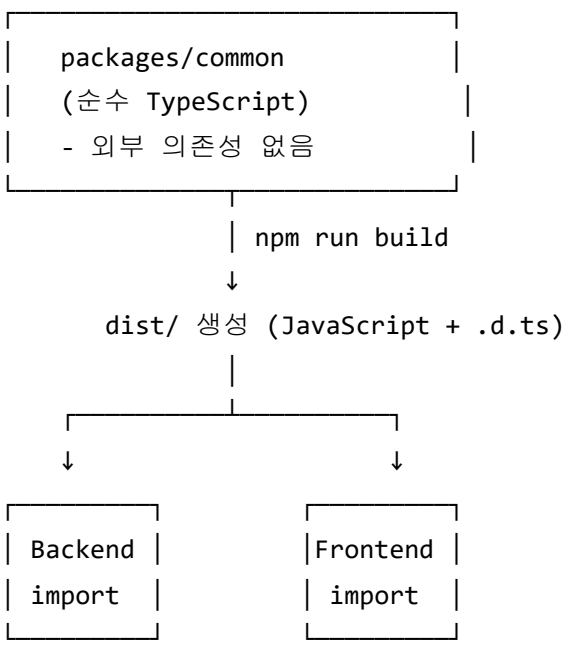
```
const ErrorMetaMap: Record<ErrorCode, ErrorMeta> = {
  [ErrorCode.TOKEN_EXPIRED]: {
    message: '세션이 만료되었습니다. 다시 로그인해주세요.',
    statusCode: 401
  },
  // ...
}
```

2.3.2.4. 권한 코드

권한 코드:

```
// 관리자 역할 코드
const CODE_SYS_ADMIN_ROLES = {
  SUPER_ADMIN: 'S-ADMIN',  // 최고 관리자
  ADMIN: 'ADMIN',          // 관리자
  EDITOR: 'EDITOR',        // 에디터
  VIEWER: 'VIEWER'         // 뷰어
}

// 공통 코드 그룹
const COMMON_CODE_GROUPS = {
  SYS_ADMIN_ROLES: 'sys_admin_roles',
  FAQ_TYPE: 'faq_type',
  QNA_TYPE: 'qna_type',
  NOTICE_TYPE: 'notice_type'
}
```

2.3.3 의존성 구조



빌드 순서:

1. `packages/common` 빌드 → `dist/` 생성
2. `be` 빌드 → `@iitp-dabt/common` import
3. `fe` 빌드 → `@iitp-dabt/common` import

2.4 기술 스택

2.4.1 Common (공통 패키지)

기술	버전	용도
TypeScript	5.8.3	타입 안정성, 타입 정의
npm	-	패키지 관리

- 외부 의존성 없음 (순수 TypeScript)
- BE/FE 모두에서 사용

2.4.2 Backend

기술	버전	용도
Runtime	Node.js 22.x	JavaScript 실행 환경
Framework	Express.js 4.18.2	웹 프레임워크
Language	TypeScript 5.8.3	타입 안전 개발
Database	PostgreSQL 12.x	관계형 데이터베이스
ORM	Sequelize 6.35.2	데이터베이스 ORM
Authentication	jsonwebtoken 9.0.2	JWT 토큰 생성/검증
Password	bcrypt 5.1.1	비밀번호 해싱
Logging	Winston 3.17.0	구조화된 로깅
	winston-daily-rotate-file 5.0.0	일별 로그 로테이션
Environment	dotenv 17.2.2	환경 변수 관리
CORS	cors 2.8.5	CORS 설정

2.4.3 Frontend

기술	버전	용도
Framework	React 18.2.0	UI 라이브러리
Language	TypeScript 5.8.3	타입 안전 개발
Build Tool	Vite 5.0.8	빌드 도구 (HMR, 빠른 빌드)
UI Library	Material-UI 5.15.0	UI 컴포넌트 라이브러리
Routing	React Router DOM 6.20.1	클라이언트 라우팅
HTTP Client	axios 1.11.0	HTTP 요청
JWT Decode	jwt-decode 4.0.0	JWT 토큰 파싱

기술	버전	용도
Styling	@emotion/react 11.11.1	CSS-in-JS

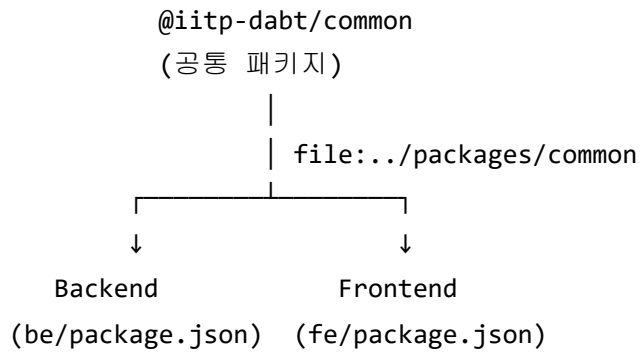
2.5 프로젝트 구조

```
05-IITP-DABT-Admin/
│
├── packages/
│   └── common/                  # 공통 패키지 (@iitp-dabt/common)
│       ├── src/
│       │   ├── types/           # 타입 정의
│       │   │   ├── api/         # API 타입
│       │   │   └── errorCodes.ts
│       │   ├── validation.ts    # 검증 함수
│       │   └── index.ts
│       ├── dist/                # 빌드 결과물
│       └── package.json
│
├── be/                          # Backend
│   ├── src/
│   │   ├── controllers/         # API 컨트롤러
│   │   ├── services/            # 비즈니스 로직
│   │   ├── repositories/        # 데이터 접근
│   │   ├── models/              # Sequelize 모델
│   │   ├── routes/              # 라우터
│   │   ├── middleware/          # 미들웨어
│   │   ├── utils/               # 유틸리티
│   │   └── index.ts             # 진입점
│   ├── dist/                    # 빌드 결과물
│   ├── logs/                    # 로그 파일
│   ├── scripts/                 # 빌드 스크립트
│   └── package.json
│
├── fe/                          # Frontend
│   ├── src/
│   │   ├── api/                 # API 호출 함수
│   │   ├── components/          # React 컴포넌트
│   │   ├── pages/               # 페이지 컴포넌트
│   │   ├── hooks/               # 커스텀 훅
│   │   ├── store/               # 상태 관리
│   │   ├── utils/               # 유틸리티
│   │   ├── routes/              # 라우팅
│   │   ├── theme/               # 테마 설정
│   │   ├── App.tsx              # 메인 앱
│   │   └── main.tsx             # 진입점
```

```
│   ├── dist/                  # 빌드 결과물
│   ├── public/                # 정적 파일
│   └── package.json
│
├── script/                     # 빌드/배포 스크립트
│   ├── build-server.js        # 서버 빌드
│   ├── deploy-server.js       # 서버 배포
│   └── ...
│
└── package.json                # 루트 (Workspace 설정)
```

2.6 패키지 간 의존성

의존성 그래프



Workspace 설정

루트 package.json:

```
{
  "workspaces": ["fe", "be", "packages/common"]
}
```

목적:

- 공통 node_modules 공유

- 의존성 중복 제거
- 일관된 버전 관리
- `npm install` 한 번으로 전체 설치

빌드 의존성

1. `packages/common` 빌드 필수
   ↓
2. `be` 빌드 (common에 의존)
   ↓
3. `fe` 빌드 (common에 의존)

빌드 명령어:

```
# 전체 빌드 (순서대로)
npm run build:common
npm run build:be
npm run build:fe

# 또는 통합 빌드
npm run build
```

3. 권한 체계 및 접근 제어

3.1 권한 체계 개요

3.1.1 2단계 권한 구조

본 시스템은 **2단계 권한 구조**를 사용합니다:

```
1차: 사용자 타입 (userType)
     ├─ 'U' (User): 일반 사용자
     └─ 'A' (Admin): 관리자
                │
                ↓
        2차: 관리자 역할 (role)
               ├─ S-ADMIN: 최고 관리자
               ├─ ADMIN: 관리자
               ├─ EDITOR: 에디터
               └─ VIEWER: 뷰어
```

3.1.2 JWT 토큰 구조

```
// 일반 사용자 토큰
{
  userId: 123,
  userType: 'U',
  iat: 1699999999,
  exp: 1700000899
}

// 관리자 토큰
{
  userId: 456,
  userType: 'A',
  role: 'S-ADMIN',  // ← 관리자인 경우만 role 포함
  iat: 1699999999,
  exp: 1700000899
}
```

3.2 관리자 역할(Role) 상세

3.2.1 S-ADMIN (최고 관리자)

역할 코드: `S-ADMIN` (CODE_SYS_ADMIN_ROLES.SUPER_ADMIN)

권한 레벨: Super Admin (최상위)

주요 권한:

- **모든 관리 기능** 접근 가능
- **운영자 계정 관리** (생성/수정/삭제/역할 변경)
- **공통 코드 관리** (시스템 설정 코드)
- **사용자 관리**
- **콘텐츠 관리** (FAQ, Q&A, 공지사항)
- **시스템 설정**

접근 가능 페이지:

- `/admin/operators` - 운영자 관리
- `/admin/code` - 공통 코드 관리
- `/admin/users` - 사용자 관리
- `/admin/faqs`, `/admin/qnas`, `/admin/notices` - 콘텐츠 관리
- 기타 모든 관리자 페이지

3.2.2 ADMIN (관리자)

역할 코드: `ADMIN`

권한 레벨: Admin

주요 권한:

- 사용자 관리 (일반 사용자 계정)
- 콘텐츠 관리 (FAQ, Q&A, 공지사항)
- OpenAPI 키 관리 (전체 사용자)

- **운영자 계정 관리** 불가
- **공통 코드 관리** 불가

접근 가능 페이지:

- `/admin/users` - 사용자 관리
- `/admin/faqs`, `/admin/qnas`, `/admin/notices` - 콘텐츠 관리
- `/admin/openapi` - OpenAPI 관리
- `/admin/operators` - 접근 불가 (메뉴 숨김)
- `/admin/code` - 접근 불가 (메뉴 숨김)

3.2.3 EDITOR (에디터)

역할 코드: `EDITOR`

권한 레벨: 편집가능 (중간)

주요 권한:

- 콘텐츠 편집 (FAQ, Q&A, 공지사항 생성/수정/삭제)
- 콘텐츠 조회
- **사용자 조회** (편집 불가)
- **사용자 생성/수정/삭제** 불가
- **운영자 관리** 불가

접근 가능 페이지:

- `/admin/faqs`, `/admin/qnas`, `/admin/notices` - 콘텐츠 편집 가능
- `/admin/users` - 조회만 (생성/수정/삭제 버튼 숨김)

3.2.4 VIEWER (뷰어)

역할 코드: `VIEWER`

권한 레벨: 조회만 가능 (최하위)

주요 권한:

- 모든 데이터 조회만 가능

- **생성/수정/삭제** 모두 불가

접근 가능 페이지:

- 모든 관리자 페이지 접근 가능
- 모든 생성/수정/삭제 버튼 숨김

3.3 권한별 기능 접근 매트릭스

3.3.1 전체 권한 매트릭스

기능 영역	세부 기능	User	VIEWER	EDITOR	ADMIN	S-ADMIN
프로필	본인 프로필 조회/수정	✓	✓	✓	✓	✓
OpenAPI	본인 키 관리	✓	-	-	-	-
	전체 키 조회	-	✓	✓	✓	✓
	전체 키 관리	-	-	-	✓	✓
FAQ	조회	✓	✓	✓	✓	✓
	생성/수정/삭제	-	-	✓	✓	✓
QNA	조회 (본인)	✓	-	-	-	-
	생성 (본인)	✓	-	-	-	-
	조회 (전체)	-	✓	✓	✓	✓
	답변 작성	-	-	✓	✓	✓
공지사항	조회	✓	✓	✓	✓	✓
	생성/수정/삭제	-	-	✓	✓	✓
사용자 관리	조회	-	✓	✓	✓	✓
	생성/수정/삭제	-	-	-	✓	✓

기능 영역	세부 기능	User	VIEWER	EDITOR	ADMIN	S-ADMIN
운영자 관리	조회	-	-	-	-	✓
	생성/수정/삭제	-	-	-	-	✓
	역할 변경	-	-	-	-	✓
코드 관리	조회	-	-	-	-	✓
	생성/수정/삭제	-	-	-	-	✓

3.4 권한 체크

3.4.1 Backend API 권한 체크

기능	Backend 체크 방식	실제 접근 가능 역할
운영자 관리	`isSAdmin()` 명시적 체크	**S-ADMIN만**
코드 관리	`checkSuperRole()` 명시적 체크	**S-ADMIN만**
사용자 관리	`isAdmin()` 체크	**모든 Admin** (S-ADMIN, ADMIN, EDITOR, VIEWER)
FAQ 관리	`adminAuthMiddleware` 만	**모든 Admin**
QNA 관리	`adminAuthMiddleware` 만	**모든 Admin**
공지사항 관리	`adminAuthMiddleware` 만	**모든 Admin**

3.4.2 Frontend UI 권한 제어

기능	Frontend 체크 함수	UI 제어
운영자 관리	`hasAccountManagementPermission()`	S-ADMIN만 메뉴/버튼 표시
코드 관리	`hasAccountManagementPermission()`	S-ADMIN만 메뉴/버튼 표시

기능	Frontend 체크 함수	UI 제어
사용자 관리	`hasUserAccountEditPermission()`	ADMIN, S-ADMIN만 버튼 표시
FAQ 관리	`hasContentEditPermission()`	EDITOR 이상만 버튼 표시
QNA 관리	`hasContentEditPermission()`	EDITOR 이상만 버튼 표시
공지사항 관리	`hasContentEditPermission()`	EDITOR 이상만 버튼 표시

3.5 Frontend 화면 접근 제어

3.5.1 라우트 가드 (Route Guard)

3.5.1.1 일반 사용자 화면 접근 제어

동작:

- 로그인 여부 확인
- 토큰 유효성 검사
- 미인증 시 → 로그인 페이지로 자동 이동

체크 로직:

1. 로컬 저장소에서 토큰 확인
2. 토큰 만료 여부 확인
3. 실패 시 로그인 페이지로 리다이렉트

적용 페이지:

- `/dashbd` - 대시보드
- `/profile` - 프로필 관리
- `/user/openapi` - OpenAPI 키 관리
- 기타 사용자 전용 페이지

3.5.1.2 관리자 화면 제어

동작:

- 관리자 로그인 여부 확인
- 관리자 타입(userType='A') 확인
- 미인증 시 → 관리자 로그인 페이지로 자동 이동

체크 로직:

1. 로컬 저장소에서 관리자 토큰 확인
2. userType이 'A'인지 확인
3. 실패 시 관리자 로그인 페이지로 리다이렉트

적용 페이지:

- `/admin/dashbd` - 관리자 대시보드
- `/admin/users` - 사용자 관리
- `/admin/faqs` - FAQ 관리
- 기타 모든 관리자 페이지

3.5.2 역할별 UI 제어

3.5.2.1 메뉴 표시/숨김

S-ADMIN 전용 메뉴:

- 운영자 관리 메뉴
- 공통 코드 관리 메뉴
- → S-ADMIN 역할만 메뉴 표시

ADMIN 이상 메뉴:

- 사용자 관리 메뉴
- → ADMIN, S-ADMIN만 메뉴 표시

EDITOR 이상 메뉴:

- 콘텐츠 관리 메뉴 (FAQ, Q&A, 공지사항)
- → EDITOR, ADMIN, S-ADMIN 메뉴 표시

모든 관리자 메뉴:

- 대시보드
- 조회 관련 메뉴
- → VIEWER 포함 모든 역할 표시

3.5.2.2 버튼/액션 활성화 제어

생성/수정/삭제 버튼:

- 콘텐츠 관련: EDITOR 이상만 표시
- 사용자 관리: ADMIN 이상만 표시
- 운영자 관리: S-ADMIN만 표시

조회 버튼:

- 모든 역할에게 표시

실제 적용 예시:

- FAQ 목록 페이지: VIEWER는 목록만 보임, EDITOR는 "생성" 버튼도 보임
- 사용자 관리 페이지: EDITOR는 목록만 보임, ADMIN은 "사용자 추가" 버튼도 보임
- 운영자 관리 페이지: S-ADMIN만 페이지 자체가 보임

3.5.3 페이지 접근 제어 Flow

```
[사용자가 페이지 접근 시도]
        ↓
[라우트 가드 체크]
 - 로그인 필요 여부 확인
 - 관리자 권한 필요 여부 확인
        ↓
미인증 → [해당 로그인 페이지로 이동]
        ↓
인증 완료 → [페이지 렌더링]
              ↓
           [역할 확인]
            - 관리자인 경우 역할(role) 추출
              ↓
           [역할별 UI 제어]
            - 메뉴 표시/숨김
            - 버튼 활성화/비활성화
            - 특정 영역 접근 제어
```

(구체적 내용은 Frontend 상세 설계서 참조)

3.6 Backend API 접근 제어

3.6.1 미들웨어 체계

3.6.1.1 기본 인증 미들웨어

기능: JWT 토큰 검증 및 사용자 정보 추출

처리 흐름:

1. 요청 헤더에서 토큰 추출
2. JWT 서명 검증
3. 토큰 만료 여부 확인
4. 사용자 정보 추출 및 저장
   - 사용자 ID
   - 사용자 타입 (User/Admin)
   - 관리자인 경우 역할(role)
5. Sliding Session 처리
   - 만료 2분 전이면 새 토큰 생성
   - 응답 헤더에 포함

에러 처리:

- 토큰 없음 → 401 에러
- 토큰 만료 → 401 에러 (로그 기록)
- 토큰 무효 → 401 에러

3.6.1.2 관리자 인증 미들웨어

기능: 관리자 권한 확인

처리 흐름:

1. 기본 인증 미들웨어 실행
2. 사용자 타입이 'A'(Admin)인지 확인
3. Admin 아니면 → 403 Forbidden
4. Admin이면 → 다음 단계 진행

적용 대상:

- /api/admin/* 모든 관리자 API

3.6.2 세부 권한 체크

3.6.2.1 Controller 단계에서 역할 체크

S-ADMIN 전용 API:

- 운영자 계정 관리
- 공통 코드 관리
- → S-ADMIN 역할 확인, 아니면 403 에러

ADMIN 이상 API:

- 사용자 관리
- → ADMIN, S-ADMIN만 허용

EDITOR 이상 API (설계 의도):

- FAQ, Q&A, 공지사항 생성/수정/삭제
- → EDITOR, ADMIN, S-ADMIN 허용 예정

모든 Admin API (현재 구현):

- FAQ, Q&A, 공지사항 관리
- → 관리자 인증만 체크 (역할 체크 없음)

(구체적 구현은 Backend 상세 설계서 참조)

3.7 권한 체크 Flow

3.7.1 전체 권한 체크 흐름

```
[Client 요청]
     ↓
[Frontend 라우트 가드]
 - PrivateRoute: isUserAuthenticated()
 - AdminProtectedRoute: isAdminAuthenticated()
     ↓
 인증 실패 → [로그인 페이지 리다이렉트]
     ↓
 인증 성공 → [페이지 렌더링]
                ↓
           [역할별 UI 제어]
             - getAdminRole()
             - hasXxxPermission()
             - 메뉴/버튼 표시 여부 결정
               ↓
           [API 호출]
             - Authorization: Bearer <token>
               ↓
           [Backend authMiddleware]
             - JWT 검증
             - userType 확인
               ↓
           User → [User API 처리]
               ↓
           Admin → [adminAuthMiddleware]
                     - userType === 'A' 확인
                       ↓
                   [Controller에서 세부 권한 체크]
                     - isSAdmin(role)       ← 운영자/코드 관리
                     - isAdmin(role)        ← 사용자 관리
                     - (현재) FAQ/QNA는 체크 없음
                       ↓
                   권한 부족 → [403 Forbidden]
                       ↓
                   권한 충족 → [Service → Repository → DB]
```

4. 전체 시스템 연동 Flow

4.1 사용자 로그인 Flow

```
[사용자]
  ↓
[/login 페이지 접속]
  ↓
[이메일/비밀번호 입력]
  ↓
[Frontend 검증]
 - isValidEmail() ← Common 패키지
 - isValidPassword() ← Common 패키지
  ↓
검증 실패 → [에러 메시지 표시]
  ↓
검증 성공 → [POST /api/auth/login]
              ↓
          [Backend authController]
           1. open_api_user 테이블 조회 (loginId)
           2. bcrypt.compare(입력 password, DB password)
           3. 비밀번호 일치 확인
              ↓
           실패 → [401 LOGIN_FAILED]
              ↓
           성공 → [JWT 토큰 생성]
                  - Access Token (15분)
                    { userId, userType: 'U', ... }
                  - Refresh Token (7일)
                  ↓
                [sys_log_user_access 기록]
                  - log_type: 'LOGIN'
                  - act_result: 'S'
                  - ip_addr, user_agent
                  ↓
                [응답]
                  {
                    result: 'ok',
                    data: {
                      accessToken: '...',
                      refreshToken: '...',
```

```
            user: { ... }
          }
        }
        ↓
[Frontend]
 1. LocalStorage에 토큰 저장
 2. 사용자 정보 저장
 3. 대시보드로 이동 (/dashbd)
```

4.2 관리자 로그인 Flow

```
[관리자]
  ↓
[/admin/login 페이지 접속]
  ↓
[이메일/비밀번호 입력]
  ↓
[Frontend 검증]
 - isValidEmail()
 - isValidPassword()
  ↓
[POST /api/auth/admin/login]
  ↓
[Backend adminAuthController]
 1. sys_adm_account 테이블 조회
    - loginId로 조회
    - status='A', del_yn='N' 확인
 2. bcrypt.compare(입력 password, DB password)
 3. 비밀번호 일치 확인
    ↓
 실패 → [401 LOGIN_FAILED]
    ↓
 성공 → [JWT 토큰 생성]
        - Access Token (15분)
          {
            userId: admId,
            userType: 'A',
            role: 'S-ADMIN' | 'ADMIN' | 'EDITOR' | 'VIEWER'
          }
        - Refresh Token (7일)
        ↓
      [sys_log_user_access 기록]
        - user_type: 'A'
        - log_type: 'LOGIN'
        - act_result: 'S'
        ↓
      [응답]
        {
          result: 'ok',
          data: {
            accessToken: '...',
```

```
            refreshToken: '...',
            admin: {
              admId: ...,
              name: '...',
              role: 'S-ADMIN',
              roleName: '최고 관리자'
            }
          }
        }
        ↓
[Frontend]
 1. LocalStorage에 admin_access_token, admin_refresh_token 저장
 2. 관리자 정보 및 role 저장
 3. 역할에 따른 메뉴 표시 결정
    - S-ADMIN → 모든 메뉴 표시
    - ADMIN → 운영자 관리/코드 관리 제외
    - EDITOR → 사용자 관리 버튼 제외
    - VIEWER → 모든 편집 버튼 숨김
 4. 관리자 대시보드로 이동 (/admin/dashbd)
```

4.3 토큰 재발급 메커니즘

본 시스템은 **2가지 방식**으로 토큰을 재발급하여 사용자 로그인 상태를 유지합니다.

4.3.1 Sliding Session (자동 갱신)

재발급 조건:

- Access Token 만료까지 **2분 미만** 남았을 때
- 사용자가 API를 호출하는 시점에 자동 감지

동작 방식:

```
[사용자가 API 호출]
  (예: 대시보드 조회, FAQ 목록 조회 등)
        ↓
[Backend에서 토큰 만료 시간 확인]
 - 현재 시각과 만료 시각 비교
 - 남은 시간 계산
        ↓
남은 시간 > 2분 → [토큰 갱신 없이 정상 처리]
        ↓
남은 시간 ≤ 2분 → [자동 토큰 갱신]
                    1. 새 Access Token 생성
                    2. 응답 헤더에 포함하여 전달
                    3. Frontend에서 자동으로 저장
                       ↓
                   [사용자는 인지 못함]
                   [로그인 상태 유지]
```

장점:

- 사용자 경험 향상 (끊김 없는 사용)
- 보안 유지 (짧은 Access Token 만료 시간)
- 별도 API 호출 불필요

4.3.2 Refresh Token (명시적 재발급)

재발급 조건:

- Access Token이 **완전히 만료**되었을 때 (15분 경과)
- Sliding Session을 놓친 경우 (사용자가 오랫동안 비활성)
- Backend가 401 에러 응답 시

동작 방식:

```
[사용자가 API 호출]
        ↓
[Backend: Access Token 만료 확인]
        ↓
[401 에러 응답]
        ↓
[Frontend에서 401 감지]
        ↓
[Refresh Token 확인]
 - LocalStorage에서 Refresh Token 추출
 - Refresh Token 유효 기간 확인 (7일)
        ↓
Refresh Token 만료 → [자동 로그아웃]
                      [로그인 페이지로 이동]
        ↓
Refresh Token 유효 → [토큰 재발급 요청]
                     1. Refresh Token 전송
                     2. 새 Access Token + 새 Refresh Token 받음
                     3. LocalStorage 업데이트
                     4. 원래 실패한 API 자동 재시도
                        ↓
                    [사용자는 인지 못함]
                    [로그인 상태 유지]
```

추가 사항:

- Refresh Token도 함께 갱신 (Rolling Refresh)
- 최대 7일간 로그인 유지 가능
- Refresh Token 만료 시에만 재로그인 필요

4.3.3 두 방식 비교 및 우선순위

구분	Sliding Session	Refresh Token
발동 조건	만료 2분 전	만료 후 (401 에러)
호출 시점	모든 API 호출 시 자동 확인	401 에러 발생 시
우선순위	1순위 (먼저 동작)	2순위 (Sliding 놓쳤을 때)
추가 API	불필요 (응답 헤더 이용)	필요 (토큰 재발급 API)

구분	Sliding Session	Refresh Token
사용자 인지	없음	없음 (자동 처리)
네트워크 부담	낮음	약간 있음 (추가 요청)

[실제 시나리오]:

시나리오 1) 활발히 사용 중 (Sliding Session)

```
로그인 → 13분 경과 → API 호출 → 자동 갱신 → 계속 사용
      → 13분 경과 → API 호출 → 자동 갱신 → 계속 사용
      (무한 반복, 7일까지)
```

시나리오 2) 오랫동안 방치 (Refresh Token)

```
로그인 → 20분 방치 → API 호출 → 401 에러 → Refresh Token 사용 → 새 토큰 발급 → 계속 사용
```

시나리오 3) 7일 경과 (재로그인 필요)

```
마지막 사용 → 7일 경과 → API 호출 → 401 에러 → Refresh Token도 만료 → 로그아웃 → 로그인 페이지
```

4.4 권한별 API 접근 Flow

4.4.1 S-ADMIN 전용 API 접근

```
[S-ADMIN 관리자]
  ↓
[POST /api/admin/admin-accounts (운영자 생성)]
  Authorization: Bearer <admin_token>
  ↓
[Backend adminAuthMiddleware]
 - userType === 'A' 확인 ✓
  ↓
[Controller: createAdminAccount]
 - getAdminRole(req) → 'S-ADMIN'
 - isSAdmin(role) 체크
  ↓
 role !== 'S-ADMIN' → [403 FORBIDDEN]
  ↓
 role === 'S-ADMIN' → [Service 실행]
                        ↓
                    [운영자 계정 생성]
                        ↓
                    [sys_log_change_his 기록]
                        ↓
                    [200 OK 응답]
```

4.4.2 일반 Admin API 접근

```
[VIEWER 관리자]
  ↓
[POST /api/admin/faq (FAQ 생성)]
  Authorization: Bearer <admin_token>
  ↓
[Backend adminAuthMiddleware]
 - userType === 'A' 확인
  ↓
[Controller: createFaqForAdmin]
 - 세부 role 체크 없음
  ↓
[Service 실행] ← VIEWER도 접근 가능 (주의!)
  ↓
[FAQ 생성 성공]
```

4.5 주요 기능 Flow

4.5.1 사용자 회원가입 Flow

```
[/register 페이지]
  ↓
[이메일, 비밀번호, 이름, 소속 입력]
  ↓
[Frontend 실시간 검증]
 - isValidEmail()
 - validatePassword() → 에러 메시지
 - isValidName()
 - isValidAffiliation()
  ↓
[POST /api/auth/register]
  {
    loginId: 'user@example.com',
    password: 'Password123!',
    name: '홍길동',
    affiliation: 'IITP'
  }
  ↓
[Backend]
 1. 이메일 중복 확인
    - open_api_user.loginId 조회
    - 이미 있으면 → 409 EMAIL_ALREADY_EXISTS
 2. Common 패키지 검증
    - isValidEmail()
    - isValidPassword()
 3. bcrypt 해싱
    - password → hashedPassword
 4. DB 저장
    - open_api_user 테이블에 INSERT
    - status='A', del_yn='N'
 5. 자동 로그인
    - JWT 토큰 생성
 6. sys_log_user_access 기록
    ↓
[응답: Access Token + Refresh Token]
  ↓
[Frontend]
```

- 토큰 저장
- 대시보드로 이동

4.5.2 OpenAPI 키 발급 Flow

```
[사용자 대시보드]
  ↓
[/user/openapi 페이지]
  ↓
[키 발급 신청 버튼 클릭]
  ↓
[키 이름, 설명, 유효기간 입력]
  ↓
[POST /api/user/openapi/keys]
  Authorization: Bearer <user_token>
  {
    keyName: '테스트 API',
    keyDesc: '개발용',
    startDt: '2024-01-01',
    endDt: '2024-12-31'
  }
  ↓
[Backend]
 1. authMiddleware: userId 추출
 2. API 키 생성
    - authKeyGenerator.generate() → 랜덤 키
    - 형식: "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
 3. DB 저장
    - open_api_auth_key 테이블에 INSERT
    - user_id, auth_key, active_yn='Y'
 4. open_api_user 업데이트
    - latest_key_created_at 갱신
 5. sys_log_change_his 기록
    ↓
[응답: { keyId, authKey }]
  ↓
[Frontend]
 - 키 목록 갱신
 - 성공 메시지 표시
 - 키 복사 기능 제공
```

4.5.3 FAQ 관리 Flow (관리자)

```
[EDITOR 관리자]
  ↓
[/admin/faqs 페이지]
  ↓
[FAQ 목록 조회]
 - hasContentEditPermission(role) → true
 - "생성" 버튼 표시 ✓
  ↓
[생성 버튼 클릭]
  ↓
[/admin/faqs/create 페이지]
  ↓
[질문, 답변, FAQ 타입 입력]
  ↓
[POST /api/admin/faq]
  Authorization: Bearer <admin_token>
  {
    question: '자주 묻는 질문',
    answer: '답변 내용',
    faqType: 'general'
  }
  ↓
[Backend]
 1. adminAuthMiddleware: userType='A' 확인 ✓
 2. Controller (현재는 role 체크 없음)
 3. Service: FAQ 생성
    - sys_faq 테이블에 INSERT
    - created_by = actorTag ('A:456')
 4. sys_log_change_his 기록
    ↓
[응답: { faqId }]
  ↓
[Frontend]
 - FAQ 목록으로 이동
 - 성공 메시지 표시
```

4.5.4 Q&A 관리 Flow

4.5.4.1 사용자: Q&A 작성

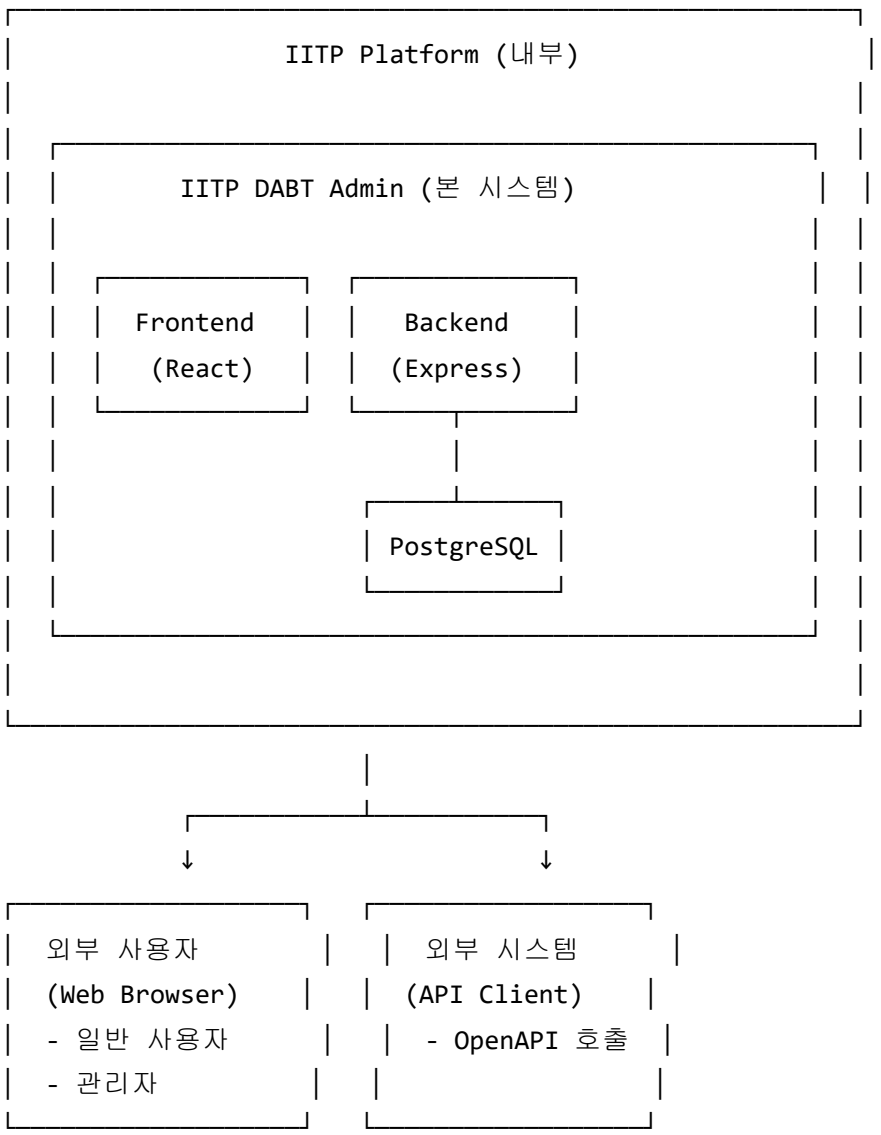
```
[일반 사용자]
  ↓
[/user/qna/create]
  ↓
[제목, 내용, 유형 입력]
  ↓
[POST /api/user/qna]
  Authorization: Bearer <user_token>
  ↓
[Backend]
 1. authMiddleware: userId 추출
 2. sys_qna 테이블에 INSERT
    - user_id, title, content
    - answered_yn='N' (미답변)
    - secret_yn (비공개 여부)
 3. sys_log_change_his 기록
    ↓
[Q&A 생성 완료]
```

4.5.4.2 관리자: Q&A 답변

```
[EDITOR 관리자]
  ↓
[/admin/qnas 목록]
 - 미답변 Q&A 조회
  ↓
[Q&A 상세 조회]
  ↓
[답변 작성 버튼]
 - hasContentEditPermission(role) → true
  ↓
[답변 내용 입력]
  ↓
[POST /api/admin/qna/:qnaId/reply]
  {
    reply: '답변 내용'
  }
  ↓
[Backend]
 1. sys_qna 업데이트
    - answer_content = reply
    - answered_yn = 'Y'
    - answered_by = actorTag
    - answered_at = NOW()
 2. sys_log_change_his 기록
    ↓
[답변 완료]
  ↓
[Frontend]
 - 목록으로 이동
 - 상태 "답변 완료"로 표시
```

4.6 내부/외부 시스템 구분

4.6.1 IITP Platform 기준 시스템 구분



4.6.2 내부 시스템 간 연동

- Frontend ↔ Backend: REST API
- Backend ↔ Database: Sequelize ORM
- 단일 시스템 (Monolithic)

5. 주요 기능 설명

5.1 사용자 기능 (User)

5.1.1 회원가입 및 로그인

회원가입:

- 이메일 기반 회원가입
- 비밀번호 규칙: 8자 이상, 영문/숫자/특수문자 포함
- 이름: 2-50자 (한글/영문/숫자/공백)
- 소속: 2-100자
- 자동 로그인 (회원가입 즉시 토큰 발급)

로그인:

- JWT 기반 인증
- Access Token: 15분
- Refresh Token: 7일
- 자동 토큰 갱신 (Sliding Session)

보안:

- 비밀번호 bcrypt 해싱 (salt rounds: 10)
- 로그인 시도 기록 (성공/실패 모두)
- IP 주소 및 User Agent 기록

5.1.2 프로필 관리

조회:

- 본인 정보 조회
- 이메일, 이름, 소속, 가입일

수정:

- 이름 수정
- 소속 수정
- 이메일 변경 불가 (로그인 ID)

비밀번호 변경:

- 현재 비밀번호 확인 필수
- 새 비밀번호 검증
- 변경 이력 기록

5.1.3 OpenAPI 키 관리

키 발급:

- 키 이름: 1-120자
- 키 설명: 1-600자 (사용 목적)
- 유효 기간 설정 (시작일, 종료일)
- 즉시 활성화 (active_yn='Y')

키 목록:

- 본인이 발급받은 키만 조회
- 키 상태 표시 (활성/비활성/만료)
- 최근 사용 일시

키 관리:

- 키 활성화/비활성화
- 키 유효기간 연장 (관리자 승인)
- 키 삭제 (논리 삭제)

5.1.4 FAQ/QNA/공지사항

FAQ 조회:

- 카테고리별 필터링
- 검색 기능
- 조회수 표시

Q&A 작성:

- 질문 작성
- 비공개 설정 가능
- 답변 알림 (답변 완료 시)

공지사항 조회:

- 중요 공지사항 상단 고정
- 게시 기간 표시
- 공개/비공개 구분

5.2 관리자 기능 (Admin)

5.2.1 콘텐츠 관리

5.2.1.1 FAQ 관리 (EDITOR 이상)

목록 조회:

- 모든 FAQ 조회 (페이징)
- 카테고리 필터
- 사용 여부 필터 (use_yn)

생성/수정/삭제:

- 질문/답변 입력
- FAQ 타입 선택 (faq_type)
- 정렬 순서 설정
- 사용 여부 설정

Frontend UI:

- `hasContentEditPermission()` 체크
- EDITOR, ADMIN, S-ADMIN만 버튼 표시

Backend API:

- `adminAuthMiddleware` 체크
- role 체크

5.2.1.2 Q&A 관리 (EDITOR 이상)

목록 조회:

- 모든 Q&A 조회
- 상태별 필터 (미답변/답변완료)
- 비공개 Q&A 포함

답변 작성:

- 답변 내용 입력
- 답변 시각 자동 기록
- 답변자 기록 (actorTag)

상태 관리:

- 미답변 → 답변 완료
- 비공개 → 공개

5.2.1.3 공지사항 관리 (EDITOR 이상)

생성:

- 제목, 내용 입력
- 공지 유형 선택 (G/S/E)
- 중요 공지사항 설정 (pinned_yn)
- 공개 여부 설정 (public_yn)
- 게시 기간 설정 (start_dt, end_dt)

수정/삭제:

- 모든 필드 수정 가능
- 삭제는 논리 삭제

5.2.2 사용자 관리 (ADMIN 이상)

목록 조회:

- 전체 사용자 조회 (페이징)
- 검색 (이메일, 이름)
- 상태 필터

사용자 생성:

- 관리자가 직접 사용자 계정 생성
- 이메일 중복 확인
- 임시 비밀번호 발급

사용자 수정:

- 이름, 소속 수정
- 상태 변경 (활성/비활성)

사용자 삭제:

- 논리 삭제 (del_yn='Y')
- 변경 이력 기록

Frontend UI:

- `hasUserAccountEditPermission()` 체크
- ADMIN, S-ADMIN만 버튼 표시

Backend API:

- `isAdmin()` 체크

5.2.3 운영자 관리 (S-ADMIN 전용)

목록 조회:

- 전체 운영자 계정 조회
- 역할별 필터
- 상태별 필터

운영자 생성:

- 이메일, 비밀번호, 이름 입력
- 역할 선택 (S-ADMIN/ADMIN/EDITOR/VIEWER)
- 소속, 설명 입력

역할 변경:

- 운영자 역할 업데이트
- 변경 이력 기록 (sys_log_change_his)

삭제:

- 논리 삭제 (del_yn='Y')

Frontend UI:

- `hasAccountManagementPermission()` 체크
- S-ADMIN만 메뉴 표시
- 다른 역할은 메뉴 자체가 안 보임

Backend API:

- `isSAdmin()` 체크 ✓
- S-ADMIN 아니면 → 403 Forbidden

5.2.4 코드 관리 (S-ADMIN 전용)

그룹 관리:

- 코드 그룹 생성
- 그룹명 수정
- 그룹 삭제

코드 관리:

- 코드 생성 (grpId, codeId, codeNm)
- 코드 수정
- 정렬 순서 변경
- 코드 삭제

계층 구조:

- 부모-자식 관계 설정 (parentCodeId)
- 다단계 계층 지원 (codeLvl)

Frontend UI:

- `hasAccountManagementPermission()` 체크
- S-ADMIN만 접근

Backend API:

- `checkSuperRole()` 체크 ✓
- S-ADMIN 아니면 → 403 Forbidden

5.3 시스템 관리

5.3.1 로그 관리

5.3.1.1 접근 로그 (sys_log_user_access)

기록 내용:

- 로그인/로그아웃 이벤트
- 사용자 ID, 사용자 타입
- 성공/실패 여부
- IP 주소, User Agent
- 에러 코드 및 메시지 (실패 시)

기록 시점:

- 로그인 시도 시 (성공/실패 모두)
- 로그아웃 시
- 토큰 만료 시

5.3.1.2 변경 로그 (sys_log_change_his)

기록 내용:

- 작업자 정보 (actor_type, actor_id)
- 액션 타입 (생성/수정/삭제)
- 대상 정보 (target_type, target_id)
- 변경 내용 요약 (chg_summary - JSONB)
- 성공/실패 여부

기록 대상:

- FAQ, Q&A, 공지사항 생성/수정/삭제
- 사용자 계정 생성/수정/삭제
- 운영자 계정 변경
- OpenAPI 키 발급/삭제

변경 내용 예시:

```
{
  "bf": { "question": "이전 질문", "answer": "이전 답변" },
  "af": { "question": "수정된 질문", "answer": "수정된 답변" }
}
```

5.3.2 모니터링

시스템 상태 확인:

- `GET /api/common/health` : 서버 상태
- `GET /api/common/version` : 버전 정보
- Database 연결 상태

API 응답 시간:

- accessLogMiddleware에서 자동 기록
- Winston 로그에 응답 시간 포함

에러 추적:

- 모든 에러 Winston 로그에 기록

- 로그 레벨: error, warn, info, debug

6. 데이터베이스 개요

6.1 주요 테이블 목록

테이블명	설명	PK	주요 용도
open_api_user	일반 사용자 계정	user_id	사용자 인증, 프로필
open_api_auth_key	OpenAPI 인증 키	key_id	API 키 관리, 인증
sys_adm_account	관리자 계정	adm_id	관리자 인증, 역할 관리
sys_common_code	공통 코드	(grp_id, code_id)	시스템 설정, 코드 관리
sys_faq	FAQ	faq_id	FAQ 관리
sys_qna	Q&A	qna_id	Q&A 관리, 답변 처리
sys_notice	공지사항	notice_id	공지사항 관리
sys_log_user_access	사용자 접근 로그	log_id	로그인/로그아웃 이력
sys_log_change_his	데이터 변경 로그	log_id	데이터 변경 이력 추적

6.2 주요 테이블 관계

```
open_api_user (사용자)
    │
    ├→ (1:N) open_api_auth_key (API 키)
    │
    └→ (1:N) sys_qna (Q&A)


sys_adm_account (관리자)
    (독립 테이블)


sys_common_code (공통 코드)
    │
    └→ (self-referencing) parent_code_id (계층 구조)


sys_log_user_access (접근 로그)
    │
    └→ user_id (사용자/관리자 ID 참조)


sys_log_change_his (변경 로그)
    │
    └→ actor_id (작업자 ID 참조)
```

6.3 핵심 테이블 개요

6.3.1 open_api_user (일반 사용자)

주요 컬럼:

- `user_id` : PK, AUTO_INCREMENT
- `login_id` : 로그인 이메일 (UNIQUE)
- `password` : bcrypt 해시 (CHAR(60))

- `user_name` : 사용자 이름
- `status` : 계정 상태 ('A': 활성)
- `del_yn` : 삭제 여부
- `affiliation` : 소속
- `latest_login_at` : 최근 로그인 시각
- `latest_key_created_at` : 최근 키 발급 시각

6.3.2 sys_adm_account (관리자)

주요 컬럼:

- `adm_id` : PK, AUTO_INCREMENT
- `login_id` : 관리자 이메일 (UNIQUE)
- `password` : bcrypt 해시
- `name` : 관리자 이름
- `roles` : **역할 코드** ('S-ADMIN', 'ADMIN', 'EDITOR', 'VIEWER')
- `status` : 계정 상태
- `del_yn` : 삭제 여부
- `affiliation` : 소속

6.3.3 sys_common_code (공통 코드)

주요 컬럼:

- `(grp_id, code_id)` : 복합 PK
- `grp_nm` : 그룹 이름
- `code_nm` : 코드 이름
- `code_type` : 'B'(서비스용), 'A'(관리자용), 'S'(시스템용)
- `parent_code_id` : 부모 코드 (계층 구조)
- `code_lvl` : 계층 레벨
- `sort_order` : 정렬 순서
- `use_yn` : 사용 여부

주요 코드 그룹:

- `sys_admin_roles` : 관리자 역할 (S-ADMIN, ADMIN, EDITOR, VIEWER)
- `faq_type` : FAQ 유형

- `qna_type` : Q&A 유형
- `notice_type` : 공지사항 유형

6.3.4 sys_log_user_access (접근 로그)

주요 컬럼:

- `log_id` : PK, BIGINT
- `user_id` : 사용자/관리자 ID
- `user_type` : 'U' | 'A'
- `log_type` : 'LOGIN', 'LOGOUT', 'LOGOUT-T-EXP' (토큰 만료)
- `act_result` : 'S'(성공), 'F'(실패)
- `err_code` , `err_msg` : 실패 시
- `ip_addr` : IP 주소
- `user_agent` : 브라우저 정보
- `access_tm` : 접근 시각

6.3.5 sys_log_change_his (변경 로그)

주요 컬럼:

- `log_id` : PK, BIGINT
- `actor_type` : 'U' | 'A'
- `actor_id` : 작업자 ID
- `action_type` : 액션 종류
- `target_type` : 대상 타입 ('FAQ', 'QNA', 'NOTICE', 'USER' 등)
- `target_id` : 대상 ID
- `act_result` : 'S' | 'F'
- `chg_summary` : 변경 내용 (JSONB)
- `act_tm` : 작업 시각

7. 환경 및 배포

7.1 개발 환경

7.1.1 로컬 개발 환경 구성

사전 요구사항:

- Node.js 22.x 이상
- npm 9.x 이상
- PostgreSQL 12.x 이상
- Git

전체 설정 (권장):

```
# 프로젝트 클론
git clone <repository-url>
cd 05-IITP-DABT-Admin

# 전체 설정 (OS 자동 감지)
npm run setup

# 개발 서버 실행
npm run dev:be    # Backend (Port 30000)
npm run dev:fe    # Frontend (Port 5173)
```

7.1.2 환경 변수

7.1.2.1 Backend 환경 변수 (.env)

필수 항목:

```
# 서버
NODE_ENV=development
PORT=30000

# 데이터베이스
DB_HOST=localhost
DB_PORT=5432
DB_NAME=iitp_dabt_admin
DB_USER=your_username
DB_PASSWORD=your_password

# JWT
JWT_SECRET=your-super-secret-jwt-key
JWT_ISSUER=iitp-dabt-api
ACCESS_TOKEN_EXPIRES_IN=15m
REFRESH_TOKEN_EXPIRES_IN=7d

# 암호화 (선택)
ENC_SECRET=your-encryption-secret

# CORS (선택)
CORS_ORIGINS=http://localhost:5173

# 로깅
LOG_LEVEL=info
```

중요:

- Backend는 **실행 시 .env 파일 필수**
- 빌드 시에는 불필요 (TypeScript 컴파일만)

7.1.2.2 Frontend 환경 변수 (.env)

선택 항목 (서브패스 배포 시만 필요):

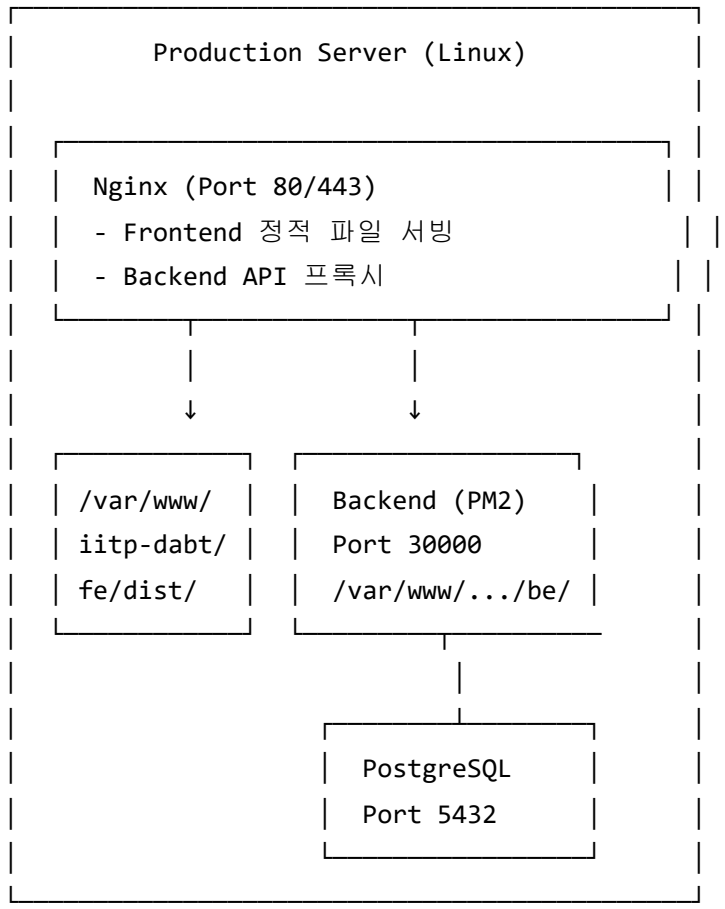
```
# 시나리오 A: 독립 도메인 (기본)
# → 설정 불필요

# 시나리오 B: 서브패스 배포
VITE_BASE=/adm/
VITE_API_BASE_URL=/adm/api
```

주의 사항:

- Frontend는 **빌드 시점**에만 환경 변수 사용
- 실행 시에는 정적 파일만 서빙 (환경 변수 미사용)

7.2 배포 구조

7.2.1 프로덕션 서버 구성



7.2.2 배포 방식

7.2.2.1. 로컬 → 서버 배포 (개발/테스트용)

```
# 로컬에서 빌드 후 서버로 전송
npm run deploy        # 전체
npm run deploy:be     # Backend만
npm run deploy:fe     # Frontend만
```

특징:

- 로컬에서 빌드

- SCP로 서버 전송
- 빠른 배포

7.2.2.1.2. 서버 간 배포 (프로덕션 권장)

빌드 서버:

```
# Git clone → 빌드 → 배포 디렉토리 준비
npm run build:server
```

실행 서버:

```
# 최초 1회: 운영 스크립트 배포
npm run deploy:server:ops

# 빌드 서버에서 파일 가져오기
npm run deploy:server

# 프로세스 시작/재시작
npm run start:server:be
npm run restart:server:fe
```

7.3 PM2 프로세스 관리

7.3.1 Backend 프로세스 관리

```
# 시작
pm2 start dist/index.js --name iitp-dabt-adm-be

# 재시작
pm2 restart iitp-dabt-adm-be

# 중지
pm2 stop iitp-dabt-adm-be

# 로그 확인
pm2 logs iitp-dabt-adm-be

# 상태 확인
pm2 status

# 재부팅 시 자동 시작
pm2 startup
pm2 save
```

7.4 Nginx 설정

7.4.1 독립 도메인 배포

```
upstream backend {
    server 127.0.0.1:30000;
}

server {
    listen 80;
    server_name admin.example.com;
    root /var/www/iitp-dabt-admin/fe/dist;
    index index.html;

    # Frontend (SPA fallback)
    location / {
        try_files $uri $uri/ /index.html;
    }

    # Backend API 프록시
    location /api/ {
        proxy_pass http://backend/api/;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        client_max_body_size 20m;
    }
}
```

7.4.2 서브패스 배포

```
server {
    listen 80;
    server_name example.com;

    # API 프록시
    location /adm/api/ {
        proxy_pass http://backend/api/;
        # ... 프록시 설정
    }

    # 정적 자산
    location ^~ /adm/assets/ {
        alias /var/www/iitp-dabt-admin/fe/dist/assets/;
        try_files $uri =404;
    }

    # 루트 레벨 정적 파일 (이미지 등)
    # 주의: 정규식 location에서 alias 사용 시 try_files와 충돌 가능하므로 제거
    location ~* ^/adm/([^/]+\.(?:png|jpg|jpeg|gif|svg|ico|woff2?|js|css|map))$ {
        alias /var/www/iitp-dabt-admin/fe/dist/$1;
        expires 7d;
        add_header Cache-Control "public, max-age=604800";
    }

    # Frontend (alias 사용)
    location /adm/ {
        alias /var/www/iitp-dabt-admin/fe/dist/;
        index index.html;
        try_files $uri $uri/ /adm/index.html;
    }
}
```

주의:

- Frontend 빌드 전 `VITE_BASE=/adm/` 설정 필수
- alias 사용 시 try_files의 fallback은 `/adm/index.html`

8. 보안

8.1 인증 및 인가

8.1.1 JWT 토큰 보안

Access Token:

- 만료 시간: 15분 (짧게 유지)
- Payload: userId, userType, role
- 서명: HS256 (JWT_SECRET)

Refresh Token:

- 만료 시간: 7일
- 재발급 전용
- httpOnly 쿠키 또는 LocalStorage

토큰 갱신:

- Sliding Session (자동 갱신)
- 만료 2분 전 새 토큰 발급

8.1.2 역할 기반 접근 제어 (RBAC)

Frontend (UI 레벨):

- 권한 체크 함수로 메뉴/버튼 제어
- 접근 불가 페이지는 리다이렉트

Backend (API 레벨):

- adminAuthMiddleware: userType 체크
- 세부 권한: Controller에서 role 체크

현재 구현:

- 운영자 관리/코드 관리: S-ADMIN 체크
- 사용자 관리: isAdmin() 체크 (모든 Admin)
- FAQ/QNA/공지사항: adminAuthMiddleware만 (role 체크 없음)

8.2 데이터 보호

8.2.1 비밀번호 보안

해싱:

- bcrypt 알고리즘
- salt rounds: 10
- 결과: CHAR(60)

검증:

- Common 패키지 `isValidPassword()` 사용
- 8자 이상, 영문/숫자/특수문자 필수

8.2.2 환경 변수 암호화

AES-256-CBC 암호화:

```
# 평문
DB_PASSWORD=mysecretpassword

# 암호화
DB_PASSWORD=ENC(aGVsbG93b3JsZA==...)
```

암호화 스크립트:

```
node be/scripts/encrypt-env.js
```

복호화 (자동):

```
import { getDecryptedEnv } from './utils/decrypt';
const dbPassword = getDecryptedEnv('DB_PASSWORD');
// ENC(...)로 시작하면 자동 복호화
```

8.2.3 CORS 설정

허용 Origin:
env 의 CORS_ORIGINS 설정에서 localhost 이외의 도메인 허용 세팅 가능합니다.

```
const corsOrigins = [
  'http://localhost:5173',  // 기본 허용
  ...process.env.CORS_ORIGINS.split(',')  // 추가 Origin
];
```

CORS 옵션:

- credentials: true (쿠키 허용)
- methods: GET, POST, PUT, DELETE
- allowedHeaders: Content-Type, Authorization

8.3 로깅 및 감사

8.3.1 Winston 로깅 (3-File Strategy)

본 시스템은 **3개의 로그 파일**로 로그를 분리하여 관리합니다.

8.3.1.1. App Log (app-YYYY-MM-DD.log)

용도: 비즈니스 로직, 일반 정보, 경고 로그

경로: `be/logs/app-YYYY-MM-DD.log`

로그 레벨: `info` 이상 (info, warn, error)

기록 내용:

- 애플리케이션 시작/종료
- 비즈니스 로직 실행 정보
- 데이터베이스 연결 상태
- 주요 이벤트 (회원가입, 로그인 성공 등)
- 경고 및 에러

사용 예시:

```
import { appLogger } from './utils/logger';

appLogger.info('[userService.ts:register] 회원가입 성공: userId=123');
appLogger.warn('[authMiddleware.ts:25] 토큰 만료 2분 전');
appLogger.error('[userController.ts:50] 에러 발생', error);
```

로그 형식:

```
[2025-11-06 10:30:45] [INFO] 회원가입 성공: userId=123
[2025-11-06 10:31:20] [WARN] 토큰 만료 2분 전
[2025-11-06 10:32:10] [ERROR] DB 연결 실패: connection timeout
```

8.3.1.2. Access Log (access-YYYY-MM-DD.log)

용도: 모든 API 요청/응답 기록

경로: `be/logs/access-YYYY-MM-DD.log`

로그 레벨: `info`

기록 내용:

- HTTP Method
- 요청 경로 (URL)
- 응답 상태 코드
- 응답 시간 (duration)
- IP 주소

- User Agent

자동 기록: `accessLogMiddleware` 에서 자동으로 모든 API 요청 기록

로그 형식:

```
[2025-11-06 10:30:45] : GET /api/user/profile 200 45ms
[2024-11-06 10:30:50] : POST /api/auth/login 200 123ms
[2024-11-06 10:31:00] : GET /api/admin/faqs 401 5ms
```

활용:

- API 사용 패턴 분석
- 성능 모니터링 (느린 API 탐지)
- 트래픽 분석

8.3.1.3. Error Log (error-YYYY-MM-DD.log)

용도: 에러 로그만 별도 저장

경로: `be/logs/error-YYYY-MM-DD.log`

로그 레벨: `error`

기록 내용:

- 에러 메시지
- 스택 트레이스 (Stack Trace)
- 에러 발생 위치
- 에러 컨텍스트 정보

로그 형식:

```
[2025-11-06 10:32:10] [ERROR] DB 연결 실패: connection timeout
Error: connect ETIMEDOUT 192.168.1.100:5432
    at TCPConnectWrap.afterConnect [as oncomplete] (net.js:1144:16)
    at Protocol._enqueue (/app/node_modules/sequelize/lib/dialects/postgres/connection-manager.j
```

활용:

- 빠른 에러 추적
- 버그 분석
- 장애 대응

8.3.2 로그 공통 설정

로그 로테이션:

- 방식: 일별 자동 로테이션
- 보관 기간: 30일
- 압축: 미사용 (빠른 조회 우선)

로그 레벨 설정:

```
# .env 파일
LOG_LEVEL=info  # 개발: debug, 운영: warn
```

레벨	설명	기록 범위
`debug`	디버그	모든 로그 (개발용)
`info`	정보	info, warn, error (기본)
`warn`	경고	warn, error (운영 권장)
`error`	에러	error만

파일 위치:

```
be/
├── logs/
│   ├── app-2024-11-06.log     # 비즈니스 로그
│   ├── access-2024-11-06.log  # API 접근 로그
│   └── error-2024-11-06.log   # 에러 전용 로그
```

8.3.3 로그 모니터링 명령어

실시간 로그 확인:

```
# App 로그
tail -f be/logs/app-$(date +%Y-%m-%d).log

# Access 로그
tail -f be/logs/access-$(date +%Y-%m-%d).log

# Error 로그
tail -f be/logs/error-$(date +%Y-%m-%d).log
```

에러 검색:

```
# 오늘 에러 전체
cat be/logs/error-$(date +%Y-%m-%d).log

# 특정 에러 검색
grep -i "database" be/logs/error-*.log

# 최근 30일 에러 건수
wc -l be/logs/error-*.log
```

API 성능 분석:

```
# 느린 API 찾기 (100ms 이상)
grep -E "[0-9]{3,}ms" be/logs/access-$(date +%Y-%m-%d).log

# 에러 응답 찾기 (4xx, 5xx)
grep -E " [45][0-9]{2} " be/logs/access-$(date +%Y-%m-%d).log
```

8.3.4 감사 로그()

기록 대상:

- 모든 로그인/로그아웃
- 데이터 생성/수정/삭제
- 권한 변경
- 민감한 작업

보관:

- 데이터베이스에 영구 저장
- 삭제 불가 (감사 목적)

9. 로그 확인

9.1 로그 확인

실시간 로그:

```
tail -f be/logs/app-$(date +%Y-%m-%d).log
```

에러 로그 필터:

```
grep -i error be/logs/app-*.log
```

9.2 성능 모니터링

API 응답 시간:

- accessLogMiddleware에서 자동 측정
- 로그에 duration 기록

데이터베이스:

- Sequelize 쿼리 로깅
- 느린 쿼리 감지

부록

Appendix A: 용어집

용어	설명
Monorepo	여러 프로젝트를 하나의 저장소에서 관리하는 구조 (Common, BE, FE 통합)
Workspace	npm의 Monorepo 관리 기능 (`npm install --workspace`)
Common Package	BE/FE에서 공유하는 공통 라이브러리 (`@iitp-dabt/common`)
JWT	JSON Web Token, 토큰 기반 인증 방식 (Access Token + Refresh Token)
Sliding Session	토큰 만료 2분 전 자동 갱신 방식 (사용자 경험 향상)
Refresh Token	Access Token 재발급용 장기 토큰 (7일)
RBAC	Role-Based Access Control, 역할 기반 접근 제어 (S-ADMIN~VIEWER)
actorTag	작업 수행자 식별자 (형식: `U:123` 또는 `A:456`)
bcrypt	비밀번호 해싱 알고리즘 (salt rounds: 10)
Paranoid Delete	Sequelize의 논리 삭제 (soft delete), `del_yn='Y'` 로 표시
Sequelize	Node.js ORM 라이브러리 (PostgreSQL 연동)
SPA	Single Page Application, 단일 페이지 애플리케이션 (React)
3-Layer Architecture	Controller → Service → Repository 아키텍처 패턴
API_URLS	Common 패키지의 API URL 상수 집합 (BE/FE 공유)
AccessLog	API 요청/응답을 자동으로 기록하는 로그 (access-*.log)
AppLog	비즈니스 로직 및 애플리케이션 이벤트 로그 (app-*.log)
ErrorLog	에러만 별도로 기록하는 로그 (error-*.log)
Winston	Node.js 로깅 라이브러리 (본 프로젝트의 로깅 시스템)
PM2	Node.js 프로세스 관리 도구 (무중단 재시작, 모니터링)

용어	설명
Nginx	웹 서버 및 리버스 프록시 (Frontend 서빙, API 프록시)
CORS	Cross-Origin Resource Sharing, 교차 출처 리소스 공유
Audit Log	감사 로그, 변경 이력 추적용 (`sys_log_change_his`)

Appendix B: 약어 풀이

약어	풀이
IITP	Institute of Information & communications Technology Planning & Evaluation
DABT	Data API Business Tool
Admin	Administrator
API	Application Programming Interface
BE	Backend
FE	Frontend
DB	Database
ORM	Object-Relational Mapping
CRUD	Create, Read, Update, Delete
FAQ	Frequently Asked Questions
QNA	Question and Answer
PK	Primary Key
FK	Foreign Key
UK	Unique Key

Appendix C: 권한 체크 함수

C.1. 권한 체크 함수 매트릭스

함수명	허용 역할	사용 위치	사용 예시
`isSAdmin(role)`	**S-ADMIN**	BE/FE	운영자 관리, 코드 관리
`isAdmin(role)`	**ALL Admin**	BE	관리자 여부 확인 (현재 사용자 관리에 사용)
`hasAccountManagementPermission(role)`	**S-ADMIN**	FE	운영자/코드 메뉴 표시 여부
`hasUserAccountEditPermission(role)`	**ADMIN+**	FE	사용자 생성/수정 버튼 표시
`hasContentEditPermission(role)`	**EDITOR+**	FE	FAQ/QNA 생성/수정 버튼 표시

C.2. Frontend 권한 체크 함수 (fe/src/utils/auth.ts)

C.2.1. 역할 확인 함수

```
// S-ADMIN 여부 확인
isSAdmin(adminRole: string | null): boolean
// 반환: adminRole === 'S-ADMIN'
// 사용: 운영자 관리, 코드 관리 메뉴 표시

// 일반 Admin 여부 확인 (모든 관리자)
isAdmin(adminRole: string | null): boolean
// 반환: ['S-ADMIN', 'ADMIN', 'EDITOR', 'VIEWER'] 중 하나
// 사용: 관리자 페이지 접근 제어
```

C.2.2. 기능별 권한 확인

```
// 콘텐츠 편집 권한 (EDITOR 이상)
hasContentEditPermission(adminRole: string | null): boolean
// 허용: S-ADMIN, ADMIN, EDITOR
// 거부: VIEWER
// 사용: FAQ, Q&A, 공지사항 생성/수정/삭제 버튼

// 사용자 계정 편집 권한 (ADMIN 이상)
hasUserAccountEditPermission(adminRole: string | null): boolean
// 허용: S-ADMIN, ADMIN
// 거부: EDITOR, VIEWER
// 사용: 사용자 계정 생성/수정/삭제 버튼

// 운영자 계정 관리 권한 (S-ADMIN만)
hasAccountManagementPermission(adminRole: string | null): boolean
// 허용: S-ADMIN
// 거부: ADMIN, EDITOR, VIEWER
// 사용: 운영자 관리, 공통 코드 관리 메뉴 표시
```

C.2.3. 사용 예시 (Frontend)

```
// 예시 1: FAQ 생성 버튼 표시 여부
const AdminFaqList = () => {
  const adminRole = getAdminRole();
  const canEdit = hasContentEditPermission(adminRole);

  return (
    <div>
      {canEdit && (
        <Button onClick={handleCreate}>FAQ 생성</Button>
      )}
    </div>
  );
};

// 예시 2: 운영자 관리 메뉴 표시 여부
const AdminSidebar = () => {
  const adminRole = getAdminRole();
  const canManageAccounts = hasAccountManagementPermission(adminRole);

  return (
    <nav>
      {canManageAccounts && (
        <MenuItem to="/admin/operators">운영자 관리</MenuItem>
      )}
    </nav>
  );
};
```

C.3. Backend 권한 체크 함수 (be/src/utils/auth.ts)

C.3.1. 역할 추출 및 확인

```
// Request에서 관리자 역할 추출
getAdminRole(req: Request): string | null
// 반환: req.user?.admRole || null
// 사용: Controller에서 권한 체크 전 역할 추출

// S-ADMIN 여부 확인
isSAdmin(adminRole: string | null): boolean
// 반환: adminRole === 'S-ADMIN'
// 사용: 운영자 계정 관리, 코드 관리 API

// 일반 Admin 여부 확인
isAdmin(adminRole: string | null): boolean
// 반환: ['S-ADMIN', 'ADMIN', 'EDITOR', 'VIEWER'] 중 하나
// 사용: 관리자 전용 API 접근 제어
```

C.3.2. 기능별 권한 확인

```
// 콘텐츠 편집 권한 (설계 의도: EDITOR 이상)
hasContentEditPermission(adminRole: string | null): boolean
// 허용: S-ADMIN, ADMIN, EDITOR
// 거부: VIEWER
// 사용: FAQ, Q&A, 공지사항 생성/수정/삭제 API (향후 추가 권장)

// 사용자 계정 편집 권한 (ADMIN 이상)
hasUserAccountEditPermission(adminRole: string | null): boolean
// 허용: S-ADMIN, ADMIN
// 거부: EDITOR, VIEWER
// 사용: 사용자 계정 관리 API (향후 추가 권장)

// 운영자 계정 관리 권한 (S-ADMIN만)
hasAccountManagementPermission(adminRole: string | null): boolean
// 허용: S-ADMIN
// 거부: ADMIN, EDITOR, VIEWER
// 사용: 운영자 계정 관리 API

// 슈퍼 관리자 체크 (통합 헬퍼)
checkSuperRole(req: Request): { adminId: number, isSuper: boolean } | null
// 반환: { adminId, isSuper: true/false } 또는 null
// 사용: 공통 코드 관리 Controller
```

C.3.3. 사용 예시 (Backend)

```
// 예시 1: 운영자 계정 생성 (S-ADMIN 전용)
export const createAdminAccount = async (req: Request, res: Response) => {
  const adminRole = getAdminRole(req);

  if (!isSAdmin(adminRole)) {
    return sendError(res, ErrorCode.FORBIDDEN, 'S-ADMIN 권한이 필요합니다.');
  }

  // 비즈니스 로직...
};

// 예시 2: FAQ 생성 (향후 개선 권장)
export const createFaqForAdmin = async (req: Request, res: Response) => {
  const adminRole = getAdminRole(req);

  // TODO: EDITOR 이상 권한 체크 추가 권장
  if (!hasContentEditPermission(adminRole)) {
    return sendError(res, ErrorCode.FORBIDDEN, 'EDITOR 이상 권한이 필요합니다.');
  }

  // 비즈니스 로직...
};

// 예시 3: 공통 코드 생성 (checkSuperRole 사용)
export const createCodeGroupByAdmin = async (req: Request, res: Response) => {
  const { adminId, isSuper } = checkSuperRole(req) || {};

  if (!isSuper) {
    return sendError(res, ErrorCode.FORBIDDEN);
  }

  // 비즈니스 로직...
};
```

Appendix D: 주요 환경 변수

D.1. Backend 환경 변수

D.1.1. 필수 환경 변수

변수명	설명	예시	비고
NODE_ENV	실행 환경	production , development	운영/개발 모드 구분
PORT	서버 포트	30000	Backend API 서버 포트
DB_HOST	DB 호스트	localhost , 192.168.1.100	PostgreSQL 서버 주소
DB_PORT	DB 포트	5432	PostgreSQL 기본 포트
DB_NAME	DB 이름	iitp_dabt_admin	데이터베이스 이름
DB_USER	DB 사용자	postgres	DB 접속 사용자
DB_PASSWORD	DB 비밀번호	(보안)	암호화 가능 (ENC(...))
JWT_SECRET	JWT 비밀키	(강력한 랜덤 문자열)	최소 32자 이상 권장

D.1.2. 선택적 환경 변수

변수명	설명	기본값	예시
JWT_ISSUER	JWT 발행자	iitp-dabt-api	토큰 검증용
ACCESS_TOKEN_EXPIRES_IN	Access Token 만료 시간	15m	30m , 1h
REFRESH_TOKEN_EXPIRES_IN	Refresh Token 만료 시간	7d	14d , 30d
ENC_SECRET	환경변수 암호화 키	-	민감 정보 암호화 시
CORS_ORIGINS	CORS 허용 Origin	localhost	https://domain.com
LOG_LEVEL	로그 레벨	info	debug , warn , error
DB_AUTO_SYNC	Sequelize 자동 동기화	false	true (개발만)

변수명	설명	기본값	예시
DB_SYNC_ALTER	Sequelize ALTER 실행	false	true (위험!)

D.2. Frontend 환경 변수

D.2.1 필수 환경 변수

변수명	설명	예시	비고
VITE_API_BASE_URL	Backend API URL	http://localhost:30000	개발 환경
		https://api.domain.com	운영 환경

D.2.2 선택적 환경 변수

변수명	설명	기본값	예시
VITE_BASE	Base URL (서브패스)	/	/adm/
VITE_API_TIMEOUT	API 타임아웃 (ms)	10000	30000
VITE_APP_TITLE	앱 타이틀	IITP DABT Admin	커스텀 가능
VITE_APP_VERSION	앱 버전	1.0.0	package.json 과 동기화

D.3. 환경 변수 파일 위치

```
프로젝트 루트/
├── be/
│   └── .env              # Backend 환경 변수
├── fe/
│   └── .env              # Frontend 환경 변수
└── packages/
    └── common/
        └── (환경 변수 없음)
```

D.4. 환경 변수 암호화 (Jasypt 스타일)

D.4.1 암호화 방법

```
# 암호화 스크립트 실행
cd be
node scripts/encrypt-env.js

# 입력 프롬프트
Enter encryption secret: your-enc-secret
Enter value to encrypt: mysecretpassword

# 출력
Encrypted: ENC(aGVsbG93b3JsZA==...)
```

D.4.2 사용 방법

1) .env 파일에 암호화된 값 설정:

```
DB_PASSWORD=ENC(aGVsbG93b3JsZA==...)
JWT_SECRET=ENC(bXlzZWNyZXRrZXk=...)
```

2) 코드에서 자동 복호화:

```
import { getDecryptedEnv } from './utils/decrypt';

const dbPassword = getDecryptedEnv('DB_PASSWORD');
// ENC(...)로 시작하면 자동 복호화
// 그렇지 않으면 원본 값 반환
```

3) 환경 변수에 암호화 키 설정:

```
export ENC_SECRET=your-enc-secret
# 또는 .env 파일에 추가
```