IITP DABT Admin

Backend 설계서

문서 버전: 1.0.0
작성일: 2025-11-12

(주)스위트케이

문서 History

버전	일자	작성자	변경 내용	비고
1.0.0	2025-11-12	(주)스위트케이	최초 작성	

목차

1. Backend 개요

1.1 Backend 역할 및 범위

IITP DABT Admin의 Backend는 시스템의 핵심 비즈니스 로직과 데이터 처리를 담당하는 API 서버입니다.

1.1.1 주요 역할

1. REST API 서버 제공

- 클라이언트(Frontend)와 데이터베이스 사이의 중재자
- RESTful 원칙을 따르는 API 엔드포인트 제공
- JSON 형식의 요청/응답 처리

2. 비즈니스 로직 처리

- 사용자 인증 및 권한 검증
- 데이터 생성, 수정, 삭제 규칙 적용
- 복잡한 비즈니스 로직 구현

3. 데이터베이스 연동

- PostgreSQL 데이터베이스와 통신
- Sequelize ORM을 통한 데이터 접근
- 트랜잭션 관리

4. 인증/인가 처리

- JWT 기반 토큰 발급 및 검증
- 역할 기반 접근 제어 (RBAC)
- Sliding Session을 통한 토큰 자동 갱신

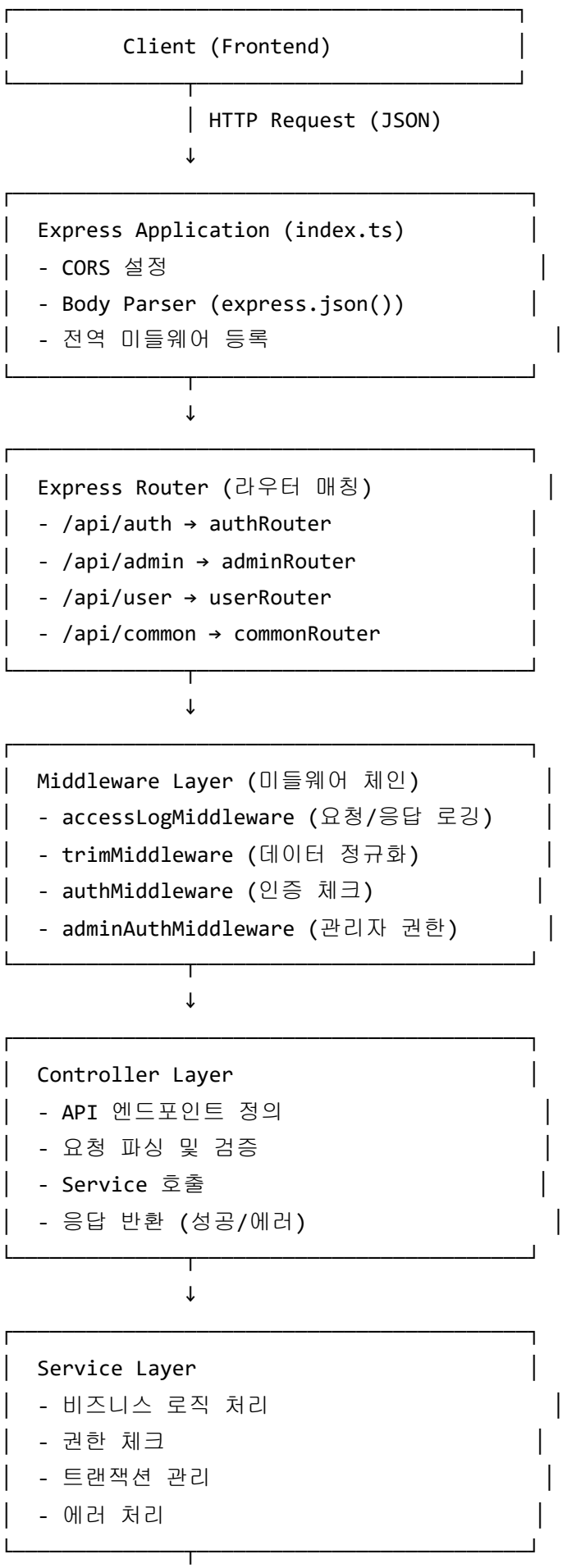
5. 로깅 및 모니터링

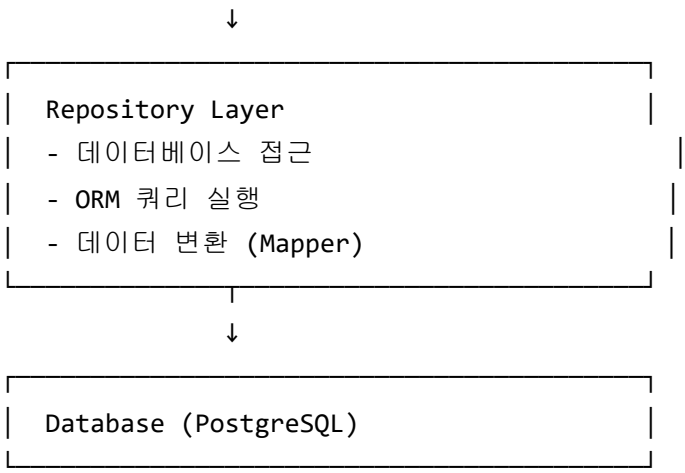
- API 접근 로그 자동 기록
- 에러 추적 및 디버깅
- 감사 로그 (Audit Log) 관리

1.2 Backend 아키텍처 개요

1.2.1 전체 아키텍처 구성

본 시스템은 **Express App → Router → Middleware → Controller → Service → Repository** 구조를 따릅니다.


Client (Frontend)
HTTP Request (JSON)
Express Application (index.ts)
- CORS 설정
- Body Parser (express.json())
- 전역 미들웨어 등록
Express Router (라우터 매칭)
- /api/auth → authRouter
- /api/admin → adminRouter
- /api/user → userRouter
- /api/common → commonRouter
Middleware Layer (미들웨어 체인)
- accessLogMiddleware (요청/응답 로깅)
- trimMiddleware (데이터 정규화)
- authMiddleware (인증 체크)
- adminAuthMiddleware (관리자 권한)
Controller Layer
- API 엔드포인트 정의
- 요청 파싱 및 검증
- Service 호출
- 응답 반환 (성공/에러)
Service Layer
- 비즈니스 로직 처리
- 권한 체크
- 트랜잭션 관리
- 에러 처리



1.3 기술 스택

1.3.1 Core Technologies

분류	기술	버전	용도
Runtime	Node.js	22.x	JavaScript 실행 환경
Framework	Express.js	4.x	웹 애플리케이션 프레임워크
Language	TypeScript	5.x	타입 안전성

1.3.2 Database

분류	기술	버전	용도
DBMS	PostgreSQL	12.x 이상	관계형 데이터베이스
ORM	Sequelize	6.x	객체-관계 매핑

1.2.3 Security & Authentication

분류	기술	용도
Authentication	jsonwebtoken	JWT 토큰 생성/검증

분류	기술	용도
Password	bcrypt	비밀번호 해싱 (salt rounds: 10)
Encryption	crypto (AES-256-CBC)	환경 변수 암호화

1.3.4 Logging & Monitoring

분류	기술	용도
Logger	Winston	로깅 시스템 (3-File Strategy)
Log Rotation	winston-daily-rotate-file	일별 로그 로테이션

1.3.5 Development Tools

분류	기술	용도
Environment	dotenv	환경 변수 관리
Build	tsc (TypeScript Compiler)	TypeScript → JavaScript 변환
Process Manager	PM2	프로세스 관리 및 무중단 재시작

1.3.6 Common Package

분류	패키지	용도
Shared	@iitp-dabt/common	BE/FE 공통 유틸리티 및 타입

1.4 참고 문서

- **IITP DABT Admin 프로젝트 아키텍처 가이드** : 프로젝트 전체 아키텍처 설명
- IITP DABT Admin Frontend 상세 설계서 : Frontend 개발 참조
- API 규격서 : API 스펙 상세
- 서버 배포 및 설치 가이드 : 서버 배포/설치/실행 가이드

2. Common 패키지 상세

2.1 Common 패키지 개요

> **전체 개요**: 프로젝트 아키텍처 가이드 2.3절 참조

패키지명: `@iitp-dabt/common`

기능: Backend와 Frontend에서 공통으로 사용하는 유틸리티 및 타입 정의

특징:

- 순수 TypeScript로 작성
- 외부 의존성 없음 (Node.js, React 독립적)
- BE/FE에서 동일한 검증 로직 및 타입 사용

중요성:

- BE/FE 간 타입 불일치 방지
- 검증 로직 중복 제거
- API 요청/응답 타입 일치 보장
- 유지보수성 향상

2.2 주요 기능

2.2.1 검증

Backend에서 요청 검증 기능입니다.

이메일 검증:

- 이메일 형식 검증 (RFC 5322 표준)
- Controller에서 로그인, 회원가입 시 사용

비밀번호 검증:

- 8자 이상, 영문/숫자/특수문자 필수
- 강도 측정 (weak, medium, strong) : FE에서 비빌번호 입력시에 검증

이름/소속 검증:

```
isValidName(name: string): boolean
isValidAffiliation(affiliation: string): boolean
```

- 길이 및 형식 검증

활용 예시 (Backend Controller):

```
import { isValidEmail, isValidPassword } from '@iitp-dabt/common';

// 회원가입 요청 검증
if (!isValidEmail(email)) {
  return sendError(res, ErrorCode.INVALID_EMAIL);
}
if (!isValidPassword(password)) {
  return sendError(res, ErrorCode.INVALID_PASSWORD);
}
```

2.2.2 타입 정의 (types/)

API 요청/응답 타입:

- Backend Controller의 요청/응답 타입
- Frontend API 호출 시 동일한 타입 사용
- 타입 안전성 보장

에러 코드:

- 시스템 전체 에러 코드 정의
- Backend에서 에러 응답 시 사용
- Frontend에서 에러 처리 시 동일 코드 체크

공통 코드 상수:

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

// 작업 유형 코드
CODE_SYS_WORK_TYPES = {
  BATCH: 'SYS-BATCH',
  MANUAL: 'SYS-MANUAL',
  USER: 'BY-USER'
}
```

활용: Backend에서 역할 체크, Frontend에서 UI 제어 시 동일 상수 사용

2.2.3 API 응답 구조

Backend의 모든 API는 통일된 응답 구조를 사용합니다.

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

성공 응답 예시:

```
{
  "result": "ok",
  "data": {
    "userId": 123,
    "name": "홍길동"
  }
}
```

에러 응답 예시:

```
{
  "result": "error",
  "errorCode": 14000,
  "message": "이메일 또는 비밀번호가 올바르지 않습니다."
}
```

2.3 에러 코드 체계

Backend의 모든 에러는 Common 패키지에 정의된 코드를 사용합니다.

2.3.1 에러 코드 범위

범위	분류	예시 코드
11xxx	기본 에러	UNKNOWN_ERROR, VALIDATION_ERROR
12xxx	요청 관련	INVALID_REQUEST, EMAIL_ALREADY_EXISTS
14xxx	인증 관련	AUTH_INVALID_CREDENTIALS, TOKEN_EXPIRED
15xxx	사용자 관련	USER_NOT_FOUND, USER_INACTIVE
16xxx	관리자 관련	ADMIN_NOT_FOUND, ADMIN_ACCESS_DENIED
17xxx	FAQ 관련	FAQ_NOT_FOUND, FAQ_CREATE_FAILED
18xxx	QNA 관련	QNA_NOT_FOUND, QNA_ALREADY_REPLIED
19xxx	공지사항 관련	NOTICE_NOT_FOUND
20xxx	OpenAPI 관련	OPENAPI_KEY_INVALID
21xxx	공통 코드 관련	COMMON_CODE_NOT_FOUND
22xxx	시스템 관련	SYS_INTERNAL_SERVER_ERROR

2.3.2 주요 에러 코드

인증 관련 (14xxx):

- `14000` : AUTH_INVALID_CREDENTIALS - 이메일 또는 비밀번호 불일치
- `14001` : TOKEN_REQUIRED - 토큰 누락
- `14002` : TOKEN_EXPIRED - 토큰 만료
- `14003` : TOKEN_INVALID - 토큰 무효
- `14004` : UNAUTHORIZED - 인증 필요

권한 관련:

- `16000` : FORBIDDEN - 권한 없음
- `16001` : ADMIN_ACCESS_DENIED - 관리자 권한 필요

사용자 관련 (15xxx):

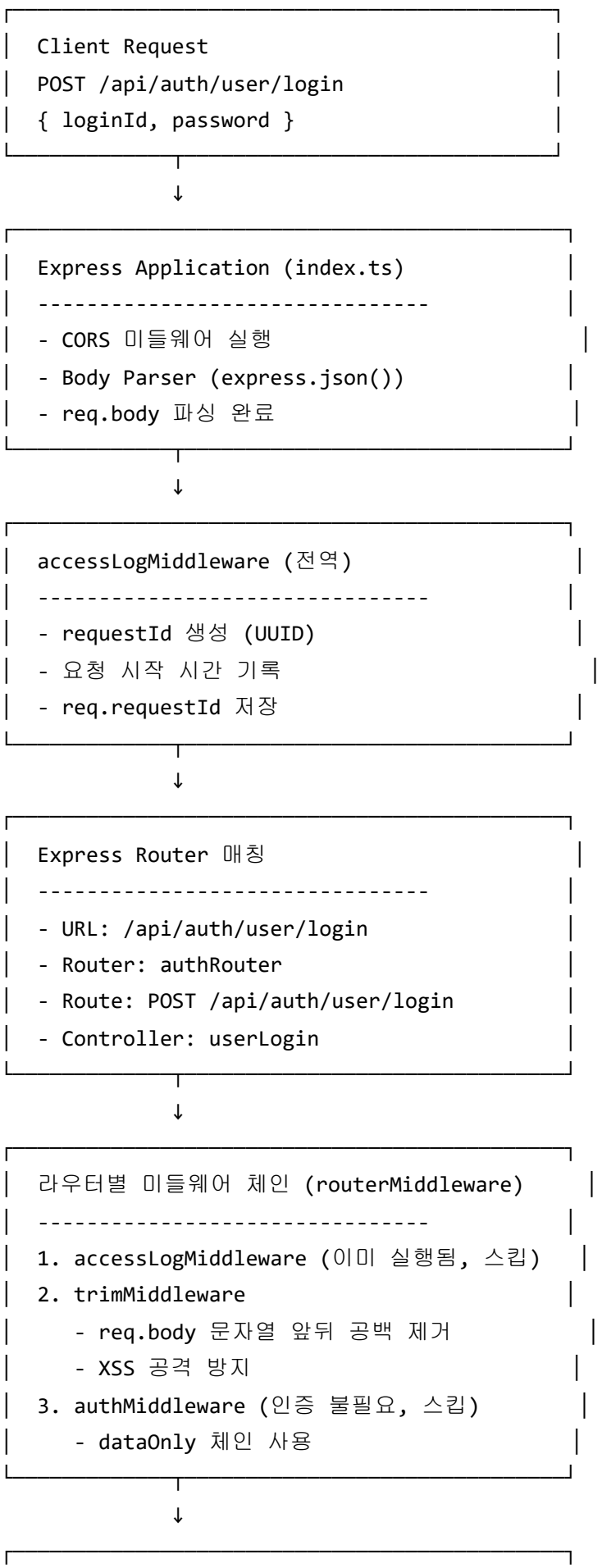
- `15000` : USER_NOT_FOUND - 사용자 없음
- `15001` : USER_INACTIVE - 비활성 계정
- `15002` : USER_DELETED - 삭제된 계정
- `12001` : EMAIL_ALREADY_EXISTS - 이메일 중복

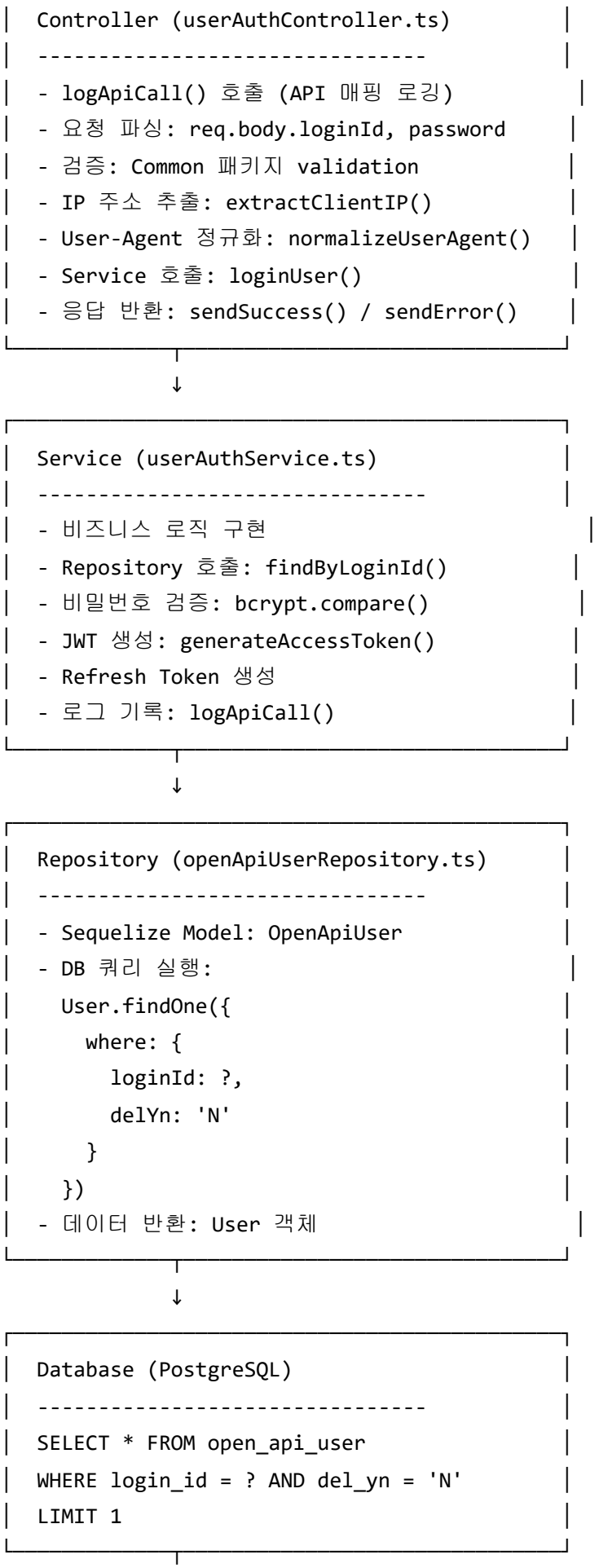
3. Backend 상세 설계

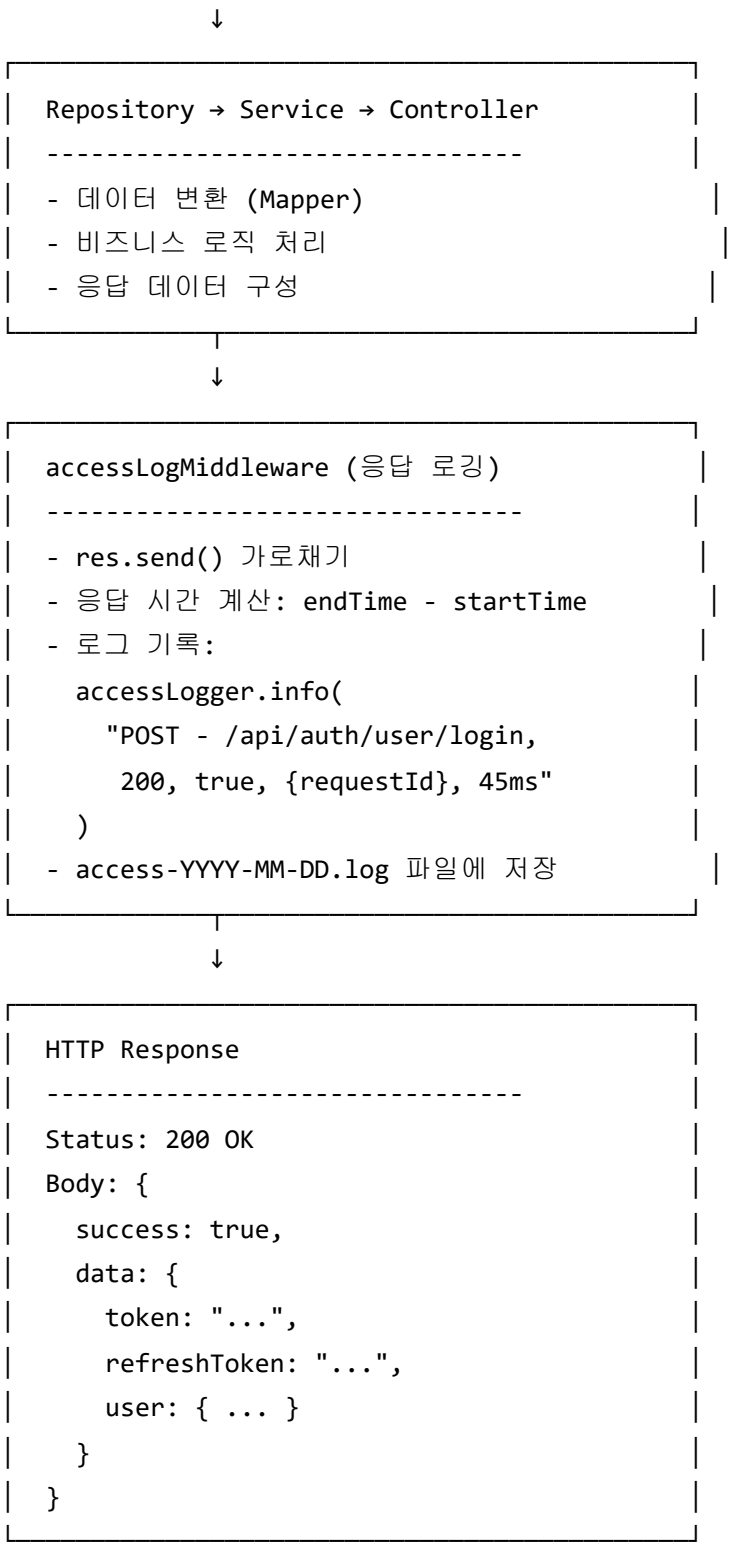
3.1 소프트웨어 아키텍처

3.1.1 전체 처리 흐름 상세

시나리오: 사용자가 `POST /api/auth/user/login` 요청을 보내는 경우


Client Request
POST /api/auth/user/login
{ loginId, password }
Express Application (index.ts)
- CORS 미들웨어 실행
- Body Parser (express.json())
- req.body 파싱 완료
accessLogMiddleware (전역)
- requestId 생성 (UUID)
- 요청 시작 시간 기록
- req.requestId 저장
Express Router 매칭
- URL: /api/auth/user/login
- Router: authRouter
- Route: POST /api/auth/user/login
- Controller: userLogin
라우터별 미들웨어 체인 (routerMiddleware)
1. accessLogMiddleware (이미 실행됨, 스킵)
2. trimMiddleware
- req.body 문자열 앞뒤 공백 제거
- XSS 공격 방지
3. authMiddleware (인증 불필요, 스킵)
- dataOnly 체인 사용


Controller (userAuthController.ts)
- logApiCall() 호출 (API 매핑 로깅)
- 요청 파싱: req.body.loginId, password
- 검증: Common 패키지 validation
- IP 주소 추출: extractClientIP()
- User-Agent 정규화: normalizeUserAgent()
- Service 호출: loginUser()
- 응답 반환: sendSuccess() / sendError()
Service (userAuthService.ts)
- 비즈니스 로직 구현
- Repository 호출: findByLoginId()
- 비밀번호 검증: bcrypt.compare()
- JWT 생성: generateAccessToken()
- Refresh Token 생성
- 로그 기록: logApiCall()
Repository (openApiUserRepository.ts)
- Sequelize Model: OpenApiUser
- DB 쿼리 실행:
User.findOne({
where: {
loginId: ?,
delYn: 'N'
}
})
- 데이터 반환: User 객체
Database (PostgreSQL)
SELECT * FROM open_api_user
WHERE login_id = ? AND del_yn = 'N'
LIMIT 1



3.1.2 미들웨어 체계

Backend의 모든 요청은 다음 미들웨어들을 거칩니다.

미들웨어 실행 순서:

1. CORS 미들웨어 (app.use(cors(...)))
2. Body Parser (express.json())
3. accessLogMiddleware (API 접근 로그 자동 기록)
4. trimMiddleware (요청 데이터 공백 제거)
5. authMiddleware / adminAuthMiddleware (인증/권한)
6. Controller 실행

각 미들웨어 설명:

1. accessLogMiddleware

- 모든 API 요청/응답 자동 기록
- `access-YYYY-MM-DD.log` 파일에 저장
- Method, URL, Status, Duration 기록

2. trimMiddleware

- 요청 데이터의 문자열 앞뒤 공백 자동 제거
- XSS 공격 방지
- 데이터 정규화

3. authMiddleware

- JWT 토큰 검증
- `req.user` 에 사용자 정보 저장
- Sliding Session 처리

4. adminAuthMiddleware

- `authMiddleware` 실행 후 추가 체크
- `userType === 'A'` 확인
- 관리자 전용 API에 적용

3.2 디렉토리 구조

```
be/
├── src/
│   ├── controllers/          # 컨트롤러 (API 엔드포인트)
│   │   ├── admin/           # 관리자 컨트롤러
│   │   │   ├── adminAccountController.ts      # 운영자 계정 관리
│   │   │   ├── adminAuthController.ts         # 관리자 인증
│   │   │   ├── adminController.ts             # 관리자 프로필
│   │   │   ├── adminFaqController.ts          # FAQ 관리
│   │   │   ├── adminNoticeController.ts       # 공지사항 관리
│   │   │   ├── adminOpenApiController.ts      # OpenAPI 키 관리
│   │   │   ├── adminQnaController.ts          # Q&A 관리
│   │   │   └── userAccountController.ts       # 사용자 계정 관리
│   │   ├── common/          # 공통 컨트롤러
│   │   │   ├── commController.ts              # 헬스체크, 버전
│   │   │   └── commonCodeController.ts        # 공통 코드
│   │   └── user/            # 사용자 컨트롤러
│   │       ├── userAuthController.ts          # 사용자 인증
│   │       ├── userController.ts              # 사용자 프로필
│   │       ├── userFaqController.ts           # FAQ 조회
│   │       ├── userNoticeController.ts        # 공지사항 조회
│   │       ├── userOpenApiController.ts       # OpenAPI 키
│   │       └── userQnaController.ts           # Q&A
│   │
│   ├── services/            # 비즈니스 로직
│   │   ├── admin/
│   │   │   ├── adminAccountService.ts
│   │   │   ├── adminAuthService.ts
│   │   │   ├── adminFaqService.ts
│   │   │   ├── adminNoticeService.ts
│   │   │   ├── adminOpenApiService.ts
│   │   │   ├── adminQnaService.ts
│   │   │   ├── adminService.ts
│   │   │   └── userAccountService.ts
│   │   ├── common/
│   │   │   └── commonCodeService.ts
│   │   └── user/
│   │       ├── userAuthService.ts
│   │       ├── userFaqService.ts
│   │       ├── userNoticeService.ts
│   │       ├── userOpenApiService.ts
```

```
│   │       ├── userQnaService.ts
│   │       └── userService.ts
│   │
│   ├── repositories/        # 데이터 접근 계층
│   │   ├── openApiAuthKeyRepository.ts
│   │   ├── openApiUserRepository.ts
│   │   ├── sysAdmAccountRepository.ts
│   │   ├── sysCommonCodeRepository.ts
│   │   ├── sysFaqRepository.ts
│   │   ├── sysLogChangeHisRepository.ts
│   │   ├── sysLogUserAccessRepository.ts
│   │   ├── sysNoticeRepository.ts
│   │   └── sysQnaRepository.ts
│   │
│   ├── models/              # Sequelize 모델 정의
│   │   ├── openApiAuthKey.ts
│   │   ├── openApiUser.ts
│   │   ├── sysAdmAccount.ts
│   │   ├── sysCommonCode.ts
│   │   ├── sysFaq.ts
│   │   ├── sysLogChangeHis.ts
│   │   ├── sysLogUserAccess.ts
│   │   ├── sysNotice.ts
│   │   ├── sysQna.ts
│   │   └── index.ts                        # 모델 초기화
│   │
│   ├── routes/              # 라우터 (API 경로 정의)
│   │   ├── adminRouter.ts                  # /api/admin/*
│   │   ├── authRouter.ts                   # /api/auth/*
│   │   ├── commonCodeRoutes.ts             # /api/common-code/*
│   │   ├── commonRouter.ts                 # /api/common/*
│   │   └── userRouter.ts                   # /api/user/*
│   │
│   ├── middleware/          # 미들웨어
│   │   ├── accessLogMiddleware.ts          # API 접근 로그
│   │   ├── authMiddleware.ts               # JWT 인증
│   │   ├── trimMiddleware.ts               # 데이터 트림
│   │   └── index.ts                        # 미들웨어 통합
│   │
│   ├── mappers/             # 데이터 변환 (DB ↔ API)
│   │   ├── commonCodeMapper.ts
│   │   ├── faqMapper.ts
│   │   ├── noticeMapper.ts
```

```
│   │   ├── openApiMapper.ts
│   │   └── qnaMapper.ts
│   │
│   ├── utils/              # 유틸리티 함수
│   │   ├── apiLogger.ts                    # API 로깅 헬퍼
│   │   ├── auth.ts                         # 권한 체크 함수
│   │   ├── authKeyGenerator.ts             # API 키 생성
│   │   ├── commonUtils.ts                  # 공통 유틸
│   │   ├── customErrors.ts                 # 커스텀 에러
│   │   ├── decrypt.ts                      # AES-256 복호화
│   │   ├── errorHandler.ts                 # 에러 핸들러
│   │   ├── jwt.ts                          # JWT 생성/검증
│   │   ├── logger.ts                       # Winston 로거
│   │   ├── queryParsers.ts                 # 쿼리 파서
│   │   ├── response.ts                     # 응답 헬퍼
│   │   ├── timeUtils.ts                    # 시간 유틸
│   │   └── trimUtils.ts                    # 트림 유틸
│   │
│   ├── types/              # TypeScript 타입 정의
│   │   └── express.d.ts                    # Express 확장 타입
│   │
│   └── index.ts            # 애플리케이션 진입점
│
├── scripts/                # 유틸리티 스크립트
│   ├── build-info.js                       # 빌드 정보
│   ├── dev-watch.js                        # 개발 모드 감시
│   ├── encrypt-env.js                      # 환경 변수 암호화
│   └── test-password-hash.js               # 비밀번호 해싱 테스트
│
├── logs/                   # 로그 파일 (자동 생성)
│   ├── app-YYYY-MM-DD.log
│   ├── access-YYYY-MM-DD.log
│   └── error-YYYY-MM-DD.log
│
├── dist/                   # 빌드 결과물 (TypeScript → JavaScript)
├── node_modules/           # 의존성
├── .env                    # 환경 변수 (Git 제외)
├── .env.example            # 환경 변수 예시
├── package.json
├── tsconfig.json
└── README.md
```

3.2.1 디렉토리별 역할

src/controllers/:

- API 엔드포인트 정의
- 요청 파싱 및 검증
- Service 호출
- 응답 반환

src/services/:

- 비즈니스 로직 구현
- 권한 체크
- Repository 호출
- 에러 처리

src/repositories/:

- 데이터베이스 접근
- Sequelize 쿼리 실행
- 데이터 반환

src/models/:

- Sequelize 모델 정의
- 테이블 스키마
- 관계 설정 (associate)

src/middleware/:

- 요청 전처리
- 인증/권한 체크
- 로깅

src/utils/:

- 공통 유틸리티 함수
- 헬퍼 함수

4. 인증 및 권한 체크

> **권한 체계 전체 개요**: 프로젝트 아키텍처 가이드 3장 참조

4.1 JWT 토큰 구조

4.1.1 Access Token Payload

```
interface JwtPayload {
  userId: number;        // 사용자 ID (user_id or adm_id)
  userType: 'U' | 'A';  // 사용자 타입 (U: User, A: Admin)
  role?: string;         // Admin인 경우 역할 코드 (S-ADMIN, ADMIN, EDITOR, VIEWER)
  iat: number;           // Issued At (발행 시각)
  exp: number;           // Expiration (만료 시각)
}
```

4.1.2 토큰 생성 예시

사용자 로그인:

```
const accessToken = jwt.sign(
  {
    userId: user.userId,
    userType: 'U'
  },
  JWT_SECRET,
  { expiresIn: '15m' }
);
```

관리자 로그인:

```
  const accessToken = jwt.sign(
    {
      userId: admin.admId,
      userType: 'A',
      role: admin.roles  // 'S-ADMIN', 'ADMIN', 'EDITOR', 'VIEWER'
    },
    JWT_SECRET,
    { expiresIn: '15m' }
  );
```

4.1.3 Refresh Token

기능: Access Token 재발급용

만료 시간: 7일

저장 위치: Frontend LocalStorage

Rolling Refresh: 토큰 재발급 시 Refresh Token도 함께 갱신

4.2 인증 미들웨어

4.2.1 authMiddleware

파일: `src/middleware/authMiddleware.ts`

기능: JWT 토큰 검증 및 사용자 정보 추출

동작 순서:

1. Authorization 헤더에서 Bearer 토큰 추출
2. JWT_SECRET으로 토큰 검증
3. Payload에서 사용자 정보 추출
4. req.user에 정보 저장
   ```
   {
     userId: number,
     userType: 'U' | 'A',
     actorTag: 'U:123' | 'A:456',
     admRole?: string
   }
   ```
5. Sliding Session 처리
   - 토큰 만료까지 2분 미만 남으면
   - 새 Access Token 생성
   - 응답 헤더에 추가 (X-New-Access-Token)
6. next() 호출

에러 처리:

- 토큰 없음 → 401 (TOKEN_REQUIRED)
- 토큰 만료 → 401 (TOKEN_EXPIRED)
- 토큰 무효 → 401 (TOKEN_INVALID)

적용 대상:

- `/api/user/*` - 사용자 전용 API
- `/api/admin/*` - 관리자 API (adminAuthMiddleware와 함께)

4.2.2 adminAuthMiddleware

파일: `src/middleware/authMiddleware.ts`

기능: 관리자 권한 확인

동작 순서:

1. authMiddleware 실행 (먼저)
2. req.user.userType === 'A' 확인
3. Admin 아니면 → 403 Forbidden
4. Admin이면 → next()

적용 대상:

- `/api/admin/*` 모든 관리자 API

특징:

- userType만 체크 (역할은 Controller에서 체크)
- 모든 Admin (S-ADMIN, ADMIN, EDITOR, VIEWER) 통과

4.3 권한 체크 (목적별 정리)

> **상세 권한 매트릭스**: 프로젝트 아키텍처 가이드 Appendix C 참조

4.3.1 운영자 계정 관리 (S-ADMIN 전용)

기능: 시스템 관리자만 다른 관리자 계정을 관리

체크 함수: `isSAdmin(adminRole)`

구현:

```
export function isSAdmin(adminRole: string | null): boolean {
  return adminRole === CODE_SYS_ADMIN_ROLES.SUPER_ADMIN;
}
```

체크 이유:

1. **권한 상승 공격 방지**: 일반 Admin이 본인을 S-ADMIN으로 승격하는 것 방지
2. **시스템 보안 핵심**: 운영자 계정 관리는 최고 권한 필요
3. **감사 추적**: 누가 어떤 역할을 부여했는지 명확히 기록

적용 API:

- `POST /api/admin/admin-accounts` - 운영자 생성
- `PUT /api/admin/admin-accounts/:id/role` - 역할 변경
- `DELETE /api/admin/admin-accounts/:id` - 운영자 삭제

체크 실패 시: `ErrorCode.FORBIDDEN` (403)

사용 예시:

```
export const createAdminAccount = async (req: Request, res: Response) => {
  const adminRole = getAdminRole(req);

  // 목적: 시스템 관리자만 다른 관리자 생성 가능
  if (!isSAdmin(adminRole)) {
    appLogger.warn(`[createAdminAccount] 권한 부족: role=${adminRole}`);
    return sendError(res, ErrorCode.FORBIDDEN, 'S-ADMIN 권한이 필요합니다.');
  }

  // 비즈니스 로직...
};
```

4.3.2 공통 코드 관리 (S-ADMIN 전용)

기능: 시스템 코드는 최고 관리자만 변경 가능

체크 함수: `checkSuperRole(req)` 또는 `isSAdmin(adminRole)`

구현:

```
export function checkSuperRole(req: Request): { adminId: number, isSuper: boolean } | null {
  const adminId = req.user?.userId;
  const adminRole = req.user?.admRole;

  if (!adminId) return null;

  return {
    adminId,
    isSuper: adminRole === CODE_SYS_ADMIN_ROLES.SUPER_ADMIN
  };
}
```

적용 API:

- `POST /api/common-code/groups` - 코드 그룹 생성
- `POST /api/common-code/codes/:grpId` - 코드 생성

- `PUT /api/common-code/codes/:grpId/:codeId` - 코드 수정
- `DELETE /api/common-code/codes/:grpId/:codeId` - 코드 삭제

체크 실패 시: `ErrorCode.FORBIDDEN` (403)

4.3.3 사용자 계정 관리 (ADMIN 이상)

기능: 일반 사용자 계정 관리는 ADMIN 이상 필요

체크 함수: `hasUserAccountEditPermission(adminRole)` (설계 의도)
현재 구현: `isAdmin(adminRole)` (모든 Admin 허용)

구현 :

```
export function isAdmin(adminRole: string | null): boolean {
  return adminRole === CODE_SYS_ADMIN_ROLES.SUPER_ADMIN ||
         adminRole === CODE_SYS_ADMIN_ROLES.ADMIN ||
         adminRole === CODE_SYS_ADMIN_ROLES.EDITOR ||
         adminRole === CODE_SYS_ADMIN_ROLES.VIEWER;
}
```

체크 이유:

1. **개인정보 보호**: 사용자 이메일, 이름 등 민감 정보 접근 제한
2. **비밀번호 초기화**: 민감한 작업은 상위 권한 필요
3. **역할 분리**: EDITOR/VIEWER는 조회만 가능해야 함 (설계 의도)

적용 API:

- `POST /api/admin/user-accounts` - 사용자 생성
- `PUT /api/admin/user-accounts/:id` - 사용자 수정
- `DELETE /api/admin/user-accounts/:id` - 사용자 삭제

4.3.4 콘텐츠 편집 (EDITOR 이상)

기능: FAQ, Q&A, 공지사항은 EDITOR 이상 편집 가능

체크 함수: `hasContentEditPermission(adminRole)` (설계 의도)

구현 :

```
export function hasContentEditPermission(adminRole: string | null): boolean {
  return adminRole === CODE_SYS_ADMIN_ROLES.SUPER_ADMIN ||
         adminRole === CODE_SYS_ADMIN_ROLES.ADMIN ||
         adminRole === CODE_SYS_ADMIN_ROLES.EDITOR;
}
```

체크 이유:

1. **역할 분리**: VIEWER는 조회만 가능
2. **콘텐츠 품질 관리**: EDITOR는 콘텐츠 관리 전담 역할
3. **작업 추적**: 누가 콘텐츠를 작성/수정했는지 명확히 기록

적용 API :

- `POST /api/admin/faqs` - FAQ 생성
- `PUT /api/admin/faqs/:id` - FAQ 수정
- `DELETE /api/admin/faqs/:id` - FAQ 삭제
- Q&A, 공지사항도 동일

4.3.5 조회 권한 (모든 Admin)

기능: 조회는 모든 관리자 가능

체크 함수: `isAdmin(adminRole)`

체크 이유:

1. **VIEWER 역할 보장**: VIEWER는 조회만 가능한 역할
2. **데이터 분석**: 모든 관리자가 데이터 조회 및 분석 가능
3. **모니터링**: 시스템 상태 확인

적용 API:

- `GET /api/admin/user-accounts` - 사용자 목록
- `GET /api/admin/faqs` - FAQ 목록
- `GET /api/admin/qnas` - Q&A 목록
- 기타 모든 조회 API

체크 실패 시: `ErrorCode.FORBIDDEN` (403)

4.3.6 권한 체크 함수 목록

함수명	허용 역할	목적	파일 위치
`isSAdmin(role)`	**S-ADMIN**	운영자/코드 관리	`utils/auth.ts`
`hasAccountManagementPermission(role)`	**S-ADMIN**	운영자 계정 관리	`utils/auth.ts`
`hasUserAccountEditPermission(role)`	**ADMIN+**	사용자 계정 편집	`utils/auth.ts`
`hasContentEditPermission(role)`	**EDITOR+**	콘텐츠 편집	`utils/auth.ts`
`isAdmin(role)`	**ALL Admin**	조회 권한	`utils/auth.ts`
`checkSuperRole(req)`	**S-ADMIN**	통합 헬퍼	`utils/commonUtils.ts`
`getAdminRole(req)`	-	역할 추출	`utils/auth.ts`

전체 함수 목록 및 사용 예시: 프로젝트 아키텍처 가이드 Appendix C

4.4 권한 체크 Flow

```
[Client Request]
  POST /api/admin/faqs
  Authorization: Bearer <token>
        ↓
[accessLogMiddleware]
  - API 접근 로그 기록 시작
        ↓
[trimMiddleware]
  - 요청 데이터 공백 제거
        ↓
[adminAuthMiddleware]
  ├→ [authMiddleware]
  │   ├ JWT 검증
  │   ├ Payload 추출
  │   ├ req.user 설정
  │   │  { userId, userType, actorTag, admRole }
  │   └ Sliding Session 체크
  │      (만료 2분 전 → 새 토큰 생성)
  │
  └→ [userType 체크]
      userType === 'A'?
      ├ Yes → next()
      └ No  → 403 Forbidden
        ↓
[Controller 실행]
  ├ getAdminRole(req)로 역할 추출
  ├ 권한 체크 함수 호출
  │  예: hasContentEditPermission(adminRole)
  │
  ├ 권한 충분?
  │  ├ Yes → Service 호출
  │  └ No  → 403 Forbidden
  │
  └ Service → Repository → DB
        ↓
[응답 반환]
  - 성공: 200 OK + data
  - 에러: 4xx/5xx + errorCode
        ↓
```

```
[accessLogMiddleware]
  - API 접근 로그 기록 완료
```

5. 주요 기능 상세

> **참고**: API 엔드포인트 및 요청/응답 스펙은 **API 규격서** 참조
> 본 섹션은 **기능 및 구현 로직** 중심으로 설명합니다.

5.1 인증 및 회원가입

5.1.1 사용자 로그인

기능: 일반 사용자가 이메일과 비밀번호로 시스템에 로그인하는 기능

처리 Flow:

```
1. 요청 검증 (Common 패키지 검증 함수)
   - isValidEmail(email)
   - 비밀번호 길이 확인
2. DB 조회 (open_api_user)
   - login_id로 사용자 조회
   - status, del_yn 확인
3. 비밀번호 비교 (bcrypt)
   - bcrypt.compare(plainPassword, hashedPassword)
4. JWT 생성 (userType: 'U')
   - Access Token (15분)
   - Refresh Token (7일)
5. 로그 기록 (sys_log_user_access)
   - log_type='LOGIN', act_result='S'
6. 응답 반환
```

DB 테이블 참조:

테이블	용도	설명
open_api_user	조회	사용자 인증 정보 확인
sys_log_user_access	생성	로그인 이력 기록

Error:

- 이메일 형식 오류 → `ErrorCode.INVALID_EMAIL` (400)
- 사용자 없음 → `ErrorCode.USER_NOT_FOUND` (404)
- 비밀번호 불일치 → `ErrorCode.AUTH_INVALID_CREDENTIALS` (401)
- 계정 비활성 → `ErrorCode.USER_INACTIVE` (403)
- 삭제된 계정 → `ErrorCode.USER_DELETED` (403)

성공 결과: Access Token + Refresh Token 발급, 로그인 상태 유지

관련 API: `POST /api/auth/login` (상세는 API 규격서 참조)

5.1.2 관리자 로그인

기능: 관리자가 이메일과 비밀번호로 시스템에 로그인하며, 역할(Role) 정보도 함께 반환됩니다

처리 Flow:

```
1. 요청 검증
2. DB 조회 (sys_adm_account)
   - login_id로 관리자 조회
   - status, del_yn 확인
3. 비밀번호 비교 (bcrypt)
4. JWT 생성 (userType: 'A', role 포함)
   - Payload에 roles 추가
5. 로그 기록 (sys_log_user_access)
   - user_type='A', log_type='LOGIN'
6. 응답 반환 (role 정보 포함)
```

DB 테이블 참조:

테이블	용도	설명
`sys_adm_account`	조회	관리자 인증 및 역할 정보 확인
`sys_log_user_access`	생성	관리자 로그인 이력 기록

Error:

- 사용자 로그인과 동일
- 추가: 관리자 없음 → `ErrorCode.ADMIN_NOT_FOUND` (404)

사용자 로그인과의 차이:

- 사용자: `role` 정보 없음
- 관리자: `role` 정보 포함 (S-ADMIN, ADMIN, EDITOR, VIEWER)

관련 API: `POST /api/auth/admin/login` (상세는 API 규격서 참조)

5.1.3 토큰 재발급

기능: Access Token이 만료되었을 때 Refresh Token을 사용하여 새 토큰을 발급받는 기능

처리 Flow:

```
1. Refresh Token 검증 (JWT)
   - 토큰 형식 확인
   - 서명 검증
2. 토큰 만료 확인 (7일 이내)
3. 새 Access Token 생성
   - 동일한 Payload 사용
4. 새 Refresh Token 생성 (Rolling Refresh)
   - 보안 강화
5. 반환
```

DB 테이블 참조:

테이블	용도	설명
없음	-	Refresh Token 검증만 (DB 접근 없음)

Error:

- Refresh Token 없음 → `ErrorCode.TOKEN_REQUIRED` (401)
- Refresh Token 만료 → `ErrorCode.TOKEN_EXPIRED` (401)
- Refresh Token 무효 → `ErrorCode.TOKEN_INVALID` (401)

Sliding Session과의 차이:

- Sliding Session: 만료 2분 전 자동 갱신 (사용자 인지 못함)
- Refresh Token: 완전 만료 후 재발급 (명시적 요청)

관련 API: `POST /api/auth/refresh` (상세는 API 규격서 참조)

참고: 프로젝트 아키텍처 가이드 섹션 4.3 - 토큰 재발급 메커니즘 전체

5.1.4 사용자 회원가입

기능: 일반 사용자가 이메일과 비밀번호로 신규 계정을 생성하는 기능

처리 Flow:

```
1. 요청 검증 (Common 패키지)
   - isValidEmail(email)
   - isValidPassword(password)
   - isValidName(name)
2. 이메일 중복 확인 (open_api_user)
3. 비밀번호 해싱 (bcrypt, salt rounds: 10)
4. DB 생성 (open_api_user)
   - status: 'ACTIVE' (기본값)
   - del_yn: 'N'
5. 응답 반환 (userId)
```

DB 테이블 참조:

테이블	용도	설명
`open_api_user`	조회	이메일 중복 확인
`open_api_user`	생성	새 사용자 계정 생성

Error:

- 이메일 형식 오류 → `ErrorCode.EMAIL_INVALID_FORMAT` (400)
- 이메일 중복 → `ErrorCode.EMAIL_ALREADY_EXISTS` (409)
- 비밀번호 검증 실패 → `ErrorCode.INVALID_PASSWORD` (400)
  - 8자 미만
  - 영문/숫자/특수문자 미포함
- 이름 검증 실패 → `ErrorCode.VALIDATION_ERROR` (400)

관련 API: `POST /api/user/register` (상세는 API 규격서 참조)

5.1.5 이메일 중복 확인 (Public)

기능: 회원가입 전 이메일 주소의 중복 여부를 확인하는 기능

처리 Flow:

1. 이메일 형식 검증 (`isValidEmail`)
2. DB 조회 (`open_api_user WHERE login_id = ?`)
3. 중복 여부 반환 (`true/false`)

DB 테이블 참조:

테이블	용도	설명
`open_api_user`	조회	이메일(login_id) 중복 확인

Error:

- 이메일 형식 오류 → `ErrorCode.EMAIL_INVALID_FORMAT` (400)

응답: `{ isAvailable: boolean }`

관련 API: `POST /api/user/email/check` (상세는 API 규격서 참조)

5.2 사용자 관리

> **권한 구분**:
>
> - **조회**: 모든 관리자 (VIEWER 포함)
> - **생성/수정/삭제**: ADMIN 이상

5.2.1 사용자 목록 조회

기능: 관리자가 전체 사용자 목록을 조회하고 검색/필터링하는 기능

권한: 모든 관리자 (VIEWER 포함)

처리 Flow:

1. 권한 체크 (isAdmin)
2. 쿼리 파라미터 파싱 (페이지, 검색어, 필터)
3. DB 조회 (open_api_user)
4. 응답 반환

DB 테이블 참조:

테이블	용도	설명
open_api_user	조회	사용자 목록 및 상태 정보 조회

쿼리 파라미터:

- page : 페이지 번호 (기본: 1)
- limit : 페이지당 개수 (기본: 10)
- search : 검색어 (이름, 이메일)
- status : 상태 필터 (ACTIVE, INACTIVE)

Error:

- 관리자 아님 → 접근 거부

관련 API: `GET /api/admin/user-accounts` (상세는 API 규격서 참조)

5.2.2 사용자 생성

기능: 관리자가 직접 일반 사용자 계정을 생성하는 기능

권한: ADMIN 이상

처리 Flow:

1. 권한 체크
2. 요청 검증 (Common 패키지)
   - isValidEmail()
   - isValidPassword()
   - isValidName()
3. 이메일 중복 확인
4. 비밀번호 해싱 (bcrypt)
5. DB 생성 (open_api_user)
6. 변경 로그 기록 (sys_log_change_his)
7. 응답 반환

DB 테이블 참조:

테이블	용도	설명
open_api_user	조회	이메일 중복 확인
open_api_user	생성	새 사용자 계정 생성
sys_log_change_his	생성	사용자 생성 이력 기록

Error:

- 이메일 형식 오류 / 중복
- 비밀번호 검증 실패

관련 API: `POST /api/admin/user-accounts` (상세는 API 규격서 참조)

5.2.3 사용자 정보 수정

기능: 사용자의 이름, 소속, 계정 상태 등을 수정하는 기능

권한: ADMIN 이상

처리 Flow:

1. 권한 체크
2. 사용자 존재 확인
3. 요청 검증
4. DB 업데이트 (open_api_user)
5. 변경 로그 기록 (before/after)
6. 응답 반환

DB 테이블 참조:

테이블	용도	설명
open_api_user	조회	수정 대상 사용자 존재 확인
open_api_user	수정	사용자 정보 업데이트
sys_log_change_his	생성	변경 전후 이력 기록 (JSONB)

Error:

- 사용자 없음
- 삭제된 사용자

관련 API: `PUT /api/admin/user-accounts/:userId` (상세는 API 규격서 참조)

5.2.4 사용자 삭제

기능: 사용자 계정을 논리적으로 삭제하는 기능 (복구 가능)

권한: ADMIN 이상

처리 Flow:

1. 권한 체크
2. 사용자 존재 확인
3. 논리 삭제 (del_yn='Y', deleted_at=NOW())
4. 변경 로그 기록
5. 응답 반환

DB 테이블 참조:

테이블	용도	설명
`open_api_user`	조회	삭제 대상 사용자 존재 확인
`open_api_user`	수정	논리 삭제 처리 (del_yn='Y')
`sys_log_change_his`	생성	삭제 이력 기록

Error:

- 사용자 없음
- 이미 삭제된 사용자

주요 특징:

- 물리 삭제 하지 않음 (Soft Delete)
- 변경 이력 기록 (감사 목적)
- 복구 가능

관련 API: `DELETE /api/admin/user-accounts/:userId` (상세는 API 규격서 참조)

5.2.5 사용자 일괄 삭제

기능: 선택한 여러 사용자 계정을 한 번에 논리 삭제하는 기능

권한: ADMIN 이상

처리 Flow:

```
1. 권한 체크
2. 사용자 ID 배열 검증 (userIds: number[])
3. 트랜잭션 시작
4. 각 사용자 존재 확인
5. 일괄 논리 삭제 (del_yn='Y' 설정)
6. 변경 로그 일괄 기록
7. 트랜잭션 커밋
8. 응답 반환 (성공/실패 개수)
```

DB 테이블 참조:

테이블	용도	설명
`open_api_user`	조회	대상 사용자들 존재 확인
`open_api_user`	수정	여러 사용자 일괄 논리 삭제
`sys_log_change_his`	생성	삭제 이력 일괄 기록

Error:

- 권한 없음 → `ErrorCode.FORBIDDEN` (403)
- 일부 사용자 없음 → 트랜잭션 롤백
- 트랜잭션 실패 → `ErrorCode.DATABASE_ERROR` (500)

주요 특징:

- **트랜잭션 보장**: 전체 성공 또는 전체 실패
- **부분 실패 처리**: 하나라도 실패하면 모두 롤백
- **실패 상세 반환**: 실패한 사용자 ID 목록 제공

관련 API: `POST /api/admin/user-accounts/list-delete` (상세는 API 규격서 참조)

5.2.6 사용자 프로필 조회

기능: 로그인한 사용자가 본인의 프로필 정보를 조회하는 기능

권한: 사용자 인증 필요

처리 Flow:

```
1. 권한 체크 (authMiddleware)
2. req.user.userId 추출
3. DB 조회 (open_api_user WHERE user_id = ?)
4. 응답 반환 (비밀번호 제외)
```

DB 테이블 참조:

테이블	용도	설명
`open_api_user`	조회	본인 프로필 정보 조회

Error:

- 인증 실패 → `ErrorCode.UNAUTHORIZED` (401)
- 사용자 없음 → `ErrorCode.USER_NOT_FOUND` (404)

응답 데이터:

- userId, email, name, affiliation, status, createdAt 등
- **제외**: password (보안)

관련 API: `GET /api/user/profile` (상세는 API 규격서 참조)

5.2.7 사용자 프로필 수정

기능: 사용자가 본인의 이름, 소속 등 프로필 정보를 수정하는 기능

권한: 사용자 인증 필요

처리 Flow:

```
1. 권한 체크
2. 요청 검증 (name, affiliation)
3. DB 업데이트 (open_api_user)
4. 변경 로그 기록 (before/after)
5. 응답 반환
```

DB 테이블 참조:

테이블	용도	설명
`open_api_user`	조회	본인 계정 확인
`open_api_user`	수정	프로필 정보 업데이트 (name, affiliation)
`sys_log_change_his`	생성	변경 이력 기록

Error:

- 인증 실패 → `ErrorCode.UNAUTHORIZED` (401)
- 필수 필드 누락 → `ErrorCode.VALIDATION_ERROR` (400)

수정 불가 필드:

- email (이메일 변경 불가)
- userId
- status (관리자만 변경 가능)

관련 API: `PUT /api/user/profile` (상세는 API 규격서 참조)

5.2.8 사용자 비밀번호 변경

기능: 사용자가 본인의 비밀번호를 변경하는 기능 (현재 비밀번호 확인 필요)

권한: 사용자 인증 필요

처리 Flow:

1. 권한 체크
2. 현재 비밀번호 검증 (bcrypt.compare)
3. 새 비밀번호 검증 (isValidPassword)
4. 새 비밀번호 해싱 (bcrypt.hash)
5. DB 업데이트 (open_api_user.password)
6. 변경 로그 기록
7. 응답 반환

DB 테이블 참조:

테이블	용도	설명
open_api_user	조회	현재 비밀번호 확인
open_api_user	수정	새 비밀번호 업데이트
sys_log_change_his	생성	비밀번호 변경 이력

Error:

- 현재 비밀번호 불일치 → `ErrorCode.AUTH_INVALID_CREDENTIALS` (401)
- 새 비밀번호 검증 실패 → `ErrorCode.INVALID_PASSWORD` (400)
- 현재 비밀번호와 동일 → `ErrorCode.SAME_AS_OLD_PASSWORD` (400)

보안:

- 현재 비밀번호 필수 확인 (타인의 비밀번호 변경 방지)

- 변경 이력 감사 로그 기록
- bcrypt 재해싱 (salt rounds: 10)

관련 API: `PUT /api/user/password` (상세는 API 규격서 참조)

5.3 운영자 관리 (S-ADMIN 전용)

> **권한 구분**:
>
> - **모든 API**: S-ADMIN만 접근 가능

5.3.1 운영자 목록 조회

기능: S-ADMIN이 전체 운영자 계정 목록을 조회하고 역할별로 필터링하는 기능

권한: S-ADMIN만

처리 Flow:

```
1. 권한 체크 (isSAdmin)
2. 쿼리 파라미터 파싱 (검색, 필터)
3. DB 조회 (sys_adm_account)
4. 응답 반환
```

DB 테이블 참조:

테이블	용도	설명
`sys_adm_account`	조회	운영자 계정 및 역할 정보 조회

쿼리 파라미터:

- `search` : 검색어 (이름, 이메일)
- `role` : 역할 필터 (S-ADMIN, ADMIN, EDITOR, VIEWER)
- `status` : 상태 필터

Error:

- S-ADMIN 권한 없음

관련 API: `GET /api/admin/admin-accounts` (상세는 API 규격서 참조)

5.3.2 운영자 생성

기능: S-ADMIN이 새 운영자 계정을 생성하고 역할을 부여하는 기능

권한: S-ADMIN만

처리 Flow:

1. 권한 체크 (isSAdmin)
2. 요청 검증 (Common 패키지)
3. 이메일 중복 확인
4. 비밀번호 해싱 (bcrypt)
5. 역할 검증 (S-ADMIN, ADMIN, EDITOR, VIEWER)
6. DB 생성 (sys_adm_account)
7. 변경 로그 기록
8. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_adm_account	조회	이메일 중복 확인
sys_adm_account	생성	새 운영자 계정 생성 (역할 포함)
sys_log_change_his	생성	운영자 생성 이력 기록

Error:

- S-ADMIN 권한 없음
- 이메일 형식 오류 / 중복
- 비밀번호 검증 실패
- 잘못된 역할 코드

주요 검증:

- 역할 코드: S-ADMIN, ADMIN, EDITOR, VIEWER 중 하나
- 이메일 중복 확인
- 비밀번호 강도 검증

관련 API: `POST /api/admin/admin-accounts` (상세는 API 규격서 참조)

5.3.3 운영자 정보 수정

기능: 운영자의 이름, 소속, 설명 등 기본 정보를 수정하는 기능 (역할 제외)

권한: S-ADMIN만

처리 Flow:

1. 권한 체크 (isSAdmin)
2. 운영자 존재 확인
3. 요청 검증
4. DB 업데이트 (sys_adm_account)
5. 변경 로그 기록 (before/after)
6. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_adm_account	조회	수정 대상 운영자 존재 확인
sys_adm_account	수정	운영자 기본 정보 업데이트
sys_log_change_his	생성	변경 전후 이력 기록

Error:

- S-ADMIN 권한 없음
- 운영자 없음
- 삭제된 운영자

관련 API: `PUT /api/admin/admin-accounts/:adminId` (상세는 API 규격서 참조)

5.3.4 운영자 역할 변경

기능: 운영자의 역할(S-ADMIN, ADMIN, EDITOR, VIEWER)을 변경하여 권한을 부여하거나 회수하는 기능

권한: S-ADMIN만

처리 Flow:

1. 권한 체크 (isSAdmin)
2. 운영자 존재 확인
3. 새 역할 검증 (S-ADMIN, ADMIN, EDITOR, VIEWER)
4. DB 업데이트 (sys_adm_account.roles)
5. 변경 로그 기록 (중요: role 변경 이력)
6. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_adm_account	조회	대상 운영자 존재 및 현재 역할 확인
sys_adm_account	수정	역할(roles) 업데이트
sys_log_change_his	생성	역할 변경 이력 기록 (감사용)

Error:

- S-ADMIN 권한 없음
- 운영자 없음
- 잘못된 역할 코드
- **본인 역할 변경 시도**

보안:

- 본인의 S-ADMIN 역할은 변경 불가
- 역할 변경 이력 감사 로그에 기록

관련 API: `PUT /api/admin/admin-accounts/:adminId/role` (상세는 API 규격서 참조)

5.3.5 운영자 삭제

기능: 운영자 계정을 논리적으로 삭제하는 기능

권한: S-ADMIN만

처리 Flow:

1. 권한 체크 (isSAdmin)
2. 운영자 존재 확인
3. 본인 삭제 방지 체크
4. 논리 삭제 (del_yn='Y')
5. 변경 로그 기록
6. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_adm_account	조회	삭제 대상 운영자 존재 확인
sys_adm_account	수정	논리 삭제 처리 (del_yn='Y')
sys_log_change_his	생성	삭제 이력 기록

Error:

- S-ADMIN 권한 없음
- 운영자 없음
- **본인 삭제 시도**
- 이미 삭제된 계정

보안: 본인 계정은 삭제 불가

관련 API: `DELETE /api/admin/admin-accounts/:adminId` (상세는 API 규격서 참조)

5.3.6 이메일 중복 확인

기능: 운영자 생성 전 이메일 주소의 중복 여부를 확인하는 기능

권한: S-ADMIN만

처리 Flow:

1. 권한 체크 (isSAdmin)
2. 이메일 형식 검증
3. DB 조회 (sys_adm_account)
4. 중복 여부 반환

DB 테이블 참조:

테이블	용도	설명
sys_adm_account	조회	이메일 중복 확인 (login_id)

Error:

- S-ADMIN 권한 없음
- 이메일 형식 오류

응답: 중복 여부 (true/false)

관련 API: `POST /api/admin/admin-accounts/check-email` (상세는 API 규격서 참조)

5.3.7 운영자 일괄 삭제

기능: 선택한 여러 운영자 계정을 한 번에 논리 삭제하는 기능

권한: S-ADMIN만

처리 Flow:

1. 권한 체크 (isSAdmin)
2. 운영자 ID 배열 검증 (adminIds: number[])
3. 본인 ID 포함 여부 체크 (포함 시 전체 거부)
4. 트랜잭션 시작
5. 각 운영자 존재 확인
6. 일괄 논리 삭제 (del_yn='Y')
7. 변경 로그 일괄 기록
8. 트랜잭션 커밋
9. 응답 반환

DB 테이블 참조:

테이블	용도	설명
`sys_adm_account`	조회	대상 운영자들 존재 확인
`sys_adm_account`	수정	여러 운영자 일괄 논리 삭제
`sys_log_change_his`	생성	삭제 이력 일괄 기록

Error:

- S-ADMIN 권한 없음 → `ErrorCode.FORBIDDEN` (403)
- **본인 ID 포함** → `ErrorCode.CANNOT_DELETE_SELF` (403)
- 일부 운영자 없음 → 트랜잭션 롤백
- 트랜잭션 실패 → `ErrorCode.DATABASE_ERROR` (500)

보안: 본인 계정 포함 시 전체 작업 거부

관련 API: `POST /api/admin/admin-accounts/list-delete` (상세는 API 규격서 참조)

5.3.8 관리자 프로필 조회

기능: 로그인한 관리자가 본인의 프로필 정보를 조회하는 기능

권한: 모든 관리자

처리 Flow:

```
1. 권한 체크 (adminAuthMiddleware)
2. req.user.userId 추출
3. DB 조회 (sys_adm_account WHERE adm_id = ?)
4. 응답 반환 (비밀번호 제외)
```

DB 테이블 참조:

테이블	용도	설명
`sys_adm_account`	조회	본인 프로필 정보 조회

Error:

- 인증 실패 → `ErrorCode.UNAUTHORIZED` (401)
- 관리자 없음 → `ErrorCode.ADMIN_NOT_FOUND` (404)

응답 데이터:

- adminId, loginId, name, affiliation, roles, status 등
- **제외**: password (보안)

관련 API: `GET /api/admin/profile` (상세는 API 규격서 참조)

5.3.9 관리자 프로필 수정

기능: 관리자가 본인의 이름, 소속 등 프로필 정보를 수정하는 기능

권한: 모든 관리자

처리 Flow:

```
1. 권한 체크
2. 요청 검증 (name, affiliation)
3. DB 업데이트 (sys_adm_account)
4. 변경 로그 기록
5. 응답 반환
```

DB 테이블 참조:

테이블	용도	설명
`sys_adm_account`	조회	본인 계정 확인
`sys_adm_account`	수정	프로필 정보 업데이트
`sys_log_change_his`	생성	변경 이력 기록

Error:

- 인증 실패 → `ErrorCode.UNAUTHORIZED` (401)
- 필수 필드 누락 → `ErrorCode.VALIDATION_ERROR` (400)

수정 불가 필드:

- loginId (이메일 변경 불가)

- roles (역할은 S-ADMIN만 변경 가능, 별도 API 사용)
- status

관련 API: `PUT /api/admin/profile` (상세는 API 규격서 참조)

5.3.10 관리자 비밀번호 변경

기능: 관리자가 본인의 비밀번호를 변경하는 기능 (현재 비밀번호 확인 필요)

권한: 모든 관리자

처리 Flow:

1. 권한 체크
2. 현재 비밀번호 검증 (bcrypt.compare)
3. 새 비밀번호 검증 (isValidPassword)
4. 새 비밀번호 해싱 (bcrypt.hash)
5. DB 업데이트
6. 변경 로그 기록
7. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_adm_account	조회	현재 비밀번호 확인
sys_adm_account	수정	새 비밀번호 업데이트
sys_log_change_his	생성	비밀번호 변경 이력

Error:

- 현재 비밀번호 불일치 → `ErrorCode.AUTH_INVALID_CREDENTIALS` (401)
- 새 비밀번호 검증 실패 → `ErrorCode.INVALID_PASSWORD` (400)
- 현재 비밀번호와 동일 → `ErrorCode.SAME_AS_OLD_PASSWORD` (400)

보안:

- 현재 비밀번호 필수 확인
- 변경 이력 감사 로그 기록

- bcrypt 재해싱 (salt rounds: 10)

관련 API: `PUT /api/admin/password` (상세는 API 규격서 참조)

5.4 콘텐츠 관리

> **권한 구분**:
>
> - **조회 (Public)**: 인증 불필요 (공개 콘텐츠)
> - **조회 (User)**: 사용자 인증 (본인 Q&A)
> - **조회 (Admin)**: 관리자 인증 (전체)
> - **생성/수정/삭제**: EDITOR 이상

5.4.1 FAQ 관리

(1) FAQ 목록 조회 (Public)

기능: 누구나 공개된 FAQ 목록을 조회할 수 있는 기능

처리 Flow:

```
1. 쿼리 파라미터 파싱
2. DB 조회 (use_yn='Y', del_yn='N')
3. 응답 반환
```

DB 테이블 참조:

테이블	용도	설명
`sys_faq`	조회	공개 FAQ 목록 조회 (사용 중인 것만)

Error: 없음

관련 API: `GET /api/faqs` (상세는 API 규격서 참조)

(2) FAQ 생성

기능: 관리자가 새로운 FAQ를 작성하는 기능

권한: EDITOR 이상

처리 Flow:

1. 권한 체크 (adminAuthMiddleware)
2. 요청 검증
3. DB 생성 (sys_faq)
4. 변경 로그 기록
5. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_faq	생성	새 FAQ 생성
sys_log_change_his	생성	FAQ 생성 이력 기록

Error:

- EDITOR 이상 권한 없음 (설계 의도)
- 필수 필드 누락

관련 API: `POST /api/admin/faqs` (상세는 API 규격서 참조)

(3) PUT /api/admin/faqs/:faqId (FAQ 수정 - Admin)

권한: `adminAuthMiddleware`

기능: 기존 FAQ 수정

처리 Flow:

1. 권한 체크
2. FAQ 존재 확인
3. 요청 검증
4. DB 업데이트 (sys_faq)
5. 변경 로그 기록
6. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_faq	조회	수정 대상 FAQ 존재 확인
sys_faq	수정	FAQ 정보 업데이트
sys_log_change_his	생성	변경 전후 이력 기록

Error:

- 권한 없음
- FAQ 없음 / 삭제된 FAQ

관련 API: `PUT /api/admin/faqs/:faqId` (상세는 API 규격서 참조)

(4) FAQ 삭제

기능: FAQ를 논리적으로 삭제하는 기능 (복구 가능)

권한: EDITOR 이상

처리 Flow:

1. 권한 체크
2. FAQ 존재 확인
3. 논리 삭제 (del_yn='Y')
4. 변경 로그 기록
5. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_faq	조회	삭제 대상 FAQ 존재 확인
sys_faq	수정	논리 삭제 처리
sys_log_change_his	생성	삭제 이력 기록

Error:

- 권한 없음
- FAQ 없음 / 이미 삭제됨

관련 API: `DELETE /api/admin/faqs/:faqId` (상세는 API 규격서 참조)

(5) FAQ 일괄 삭제

기능: 선택한 여러 FAQ를 한 번에 논리 삭제하는 기능

권한: EDITOR 이상

처리 Flow:

1. 권한 체크
2. FAQ ID 배열 검증 (faqIds: number[])
3. 트랜잭션 시작
4. 각 FAQ 존재 확인
5. 일괄 논리 삭제
6. 변경 로그 일괄 기록
7. 트랜잭션 커밋
8. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_faq	조회	대상 FAQ들 존재 확인
sys_faq	수정	여러 FAQ 일괄 논리 삭제
sys_log_change_his	생성	삭제 이력 일괄 기록

Error:

- 권한 없음
- 일부 FAQ 없음 (트랜잭션 롤백)
- 트랜잭션 실패

주요 특징: 트랜잭션으로 전체 성공 또는 전체 실패

관련 API: `POST /api/admin/faqs/list-delete` (상세는 API 규격서 참조)

5.4.2 Q&A 관리

(1) 본인 Q&A 조회 (User)

기능: 일반 사용자가 본인이 작성한 Q&A 목록을 조회하는 기능

권한: 사용자 인증 필요

처리 Flow:

1. 권한 체크 (authMiddleware)
2. req.user.userId 추출
3. DB 조회 (sys_qna WHERE user_id = ?)
4. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_qna	조회	본인 Q&A 목록 조회

Error:

- 인증 실패 → `ErrorCode.UNAUTHORIZED` (401)

(2) POST /api/user/qnas (Q&A 작성 - User)

권한: `authMiddleware` → User 인증

기능: 사용자가 Q&A 질문 작성

처리 Flow:

1. 권한 체크
2. 요청 검증
3. DB 생성 (sys_qna, user_id=req.user.userId)
4. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_qna	생성	새 Q&A 질문 생성 (사용자 작성)

Error:

- 인증 실패
- 필수 필드 누락

관련 API: `POST /api/user/qnas` (상세는 API 규격서 참조)

(3) 전체 Q&A 조회 (Admin)

기능: 관리자가 모든 사용자의 Q&A를 조회하는 기능 (비공개 Q&A 포함)

권한: 모든 관리자

처리 Flow:

1. 권한 체크 (adminAuthMiddleware)
2. 쿼리 파라미터 파싱
3. DB 조회 (sys_qna 전체)
4. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_qna	조회	전체 Q&A 목록 (비공개 포함)

쿼리 파라미터:

- `status` : 상태 필터 (미답변/답변완료)
- `search` : 검색어

관련 API: `GET /api/admin/qnas` (상세는 API 규격서 참조)

(4) Q&A 답변 작성

기능: 관리자가 사용자의 질문에 답변을 작성하는 기능

권한: EDITOR 이상

처리 Flow:

1. 권한 체크
2. Q&A 존재 확인
3. 답변 내용 검증
4. DB 업데이트 (sys_qna.answer, answered_at)
5. 변경 로그 기록
6. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_qna	조회	답변 대상 Q&A 존재 확인
sys_qna	수정	답변 내용 및 답변 시각 업데이트
sys_log_change_his	생성	답변 이력 기록

Error:

- 권한 없음
- Q&A 없음
- 이미 답변된 Q&A

주요 처리:

- 답변 내용 및 답변 시각 업데이트

- 상태를 '답변완료'로 변경

관련 API: `POST /api/admin/qnas/:qnaId/answer` (상세는 API 규격서 참조)

(5) Q&A 수정

기능: Q&A의 질문 또는 답변 내용을 수정하는 기능

권한: EDITOR 이상

DB 테이블 참조:

테이블	용도	설명
sys_qna	조회	수정 대상 Q&A 존재 확인
sys_qna	수정	Q&A 정보 업데이트
sys_log_change_his	생성	변경 이력 기록

Error:

- 권한 없음
- Q&A 없음

관련 API: `PUT /api/admin/qnas/:qnaId` (상세는 API 규격서 참조)

(6) Q&A 삭제

기능: Q&A를 논리적으로 삭제하는 기능 (복구 가능)

권한: EDITOR 이상

DB 테이블 참조:

테이블	용도	설명
sys_qna	조회	삭제 대상 Q&A 존재 확인
sys_qna	수정	논리 삭제 처리

테이블	용도	설명
sys_log_change_his	생성	삭제 이력 기록

Error:

- 권한 없음
- Q&A 없음

관련 API: `DELETE /api/admin/qnas/:qnaId` (상세는 API 규격서 참조)

(7) Q&A 일괄 삭제

기능: 선택한 여러 Q&A를 한 번에 논리 삭제하는 기능

권한: EDITOR 이상

처리 Flow:

1. 권한 체크
2. Q&A ID 배열 검증 (qnaIds: number[])
3. 트랜잭션 시작
4. 각 Q&A 존재 확인
5. 일괄 논리 삭제
6. 변경 로그 일괄 기록
7. 트랜잭션 커밋
8. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_qna	조회	대상 Q&A들 존재 확인
sys_qna	수정	여러 Q&A 일괄 논리 삭제
sys_log_change_his	생성	삭제 이력 일괄 기록

Error:

- 권한 없음

- 일부 Q&A 없음 (트랜잭션 롤백)
- 트랜잭션 실패

관련 API: `POST /api/admin/qnas/list-delete` (상세는 API 규격서 참조)

5.4.3 공지사항 관리

(1) 공지사항 목록 조회 (Public)

기능: 누구나 공개된 공지사항 목록을 조회하는 기능 (중요 공지사항 상단 고정)

권한: Public

처리 Flow:

```
1. 쿼리 파라미터 파싱
2. DB 조회 (public_yn='Y', 게시 기간 내)
3. 응답 반환 (pinned_yn='Y' 우선)
```

DB 테이블 참조:

테이블	용도	설명
`sys_notice`	조회	공개 공지사항 조회 (게시 중인 것만)

주요 특징:

- 중요 공지사항 (`pinned_yn='Y'`) 우선 표시
- 게시 기간 내의 공지사항만 조회

관련 API: `GET /api/notices` (상세는 API 규격서 참조)

(2) 공지사항 생성

기능: 관리자가 새 공지사항을 작성하고 게시 기간을 설정하는 기능

권한: EDITOR 이상

처리 Flow:

1. 권한 체크
2. 요청 검증
3. 게시 기간 검증 (start_dt < end_dt)
4. DB 생성 (sys_notice)
5. 변경 로그 기록
6. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_notice	생성	새 공지사항 생성
sys_log_change_his	생성	공지사항 생성 이력 기록

Error:

- 관리자 아님 → ErrorCode.FORBIDDEN (403)
- 필수 필드 누락 → ErrorCode.VALIDATION_ERROR (400)
- 게시 기간 오류 → ErrorCode.INVALID_DATE_RANGE (400)

(3) PUT /api/admin/notices/:noticeId (공지사항 수정 - Admin)

권한: adminAuthMiddleware

기능: 기존 공지사항 수정

DB 테이블 참조:

테이블	용도	설명
sys_notice	조회	수정 대상 공지사항 존재 확인
sys_notice	수정	공지사항 정보 업데이트
sys_log_change_his	생성	변경 전후 이력 기록

Error:

- 권한 없음
- 공지사항 없음

- 게시 기간 오류

관련 API: `PUT /api/admin/notices/:noticeId` (상세는 API 규격서 참조)

(4) 공지사항 삭제

기능: 공지사항을 논리적으로 삭제하는 기능 (복구 가능)

권한: EDITOR 이상

DB 테이블 참조:

테이블	용도	설명
sys_notice	조회	삭제 대상 공지사항 존재 확인
sys_notice	수정	논리 삭제 처리
sys_log_change_his	생성	삭제 이력 기록

Error:

- 권한 없음
- 공지사항 없음

관련 API: `DELETE /api/admin/notices/:noticeId` (상세는 API 규격서 참조)

(5) 공지사항 일괄 삭제

기능: 선택한 여러 공지사항을 한 번에 논리 삭제하는 기능

권한: EDITOR 이상

처리 Flow:

1. 권한 체크
2. 공지사항 ID 배열 검증 (noticeIds: number[])
3. 트랜잭션 시작
4. 각 공지사항 존재 확인
5. 일괄 논리 삭제
6. 변경 로그 일괄 기록
7. 트랜잭션 커밋
8. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_notice	조회	대상 공지사항들 존재 확인
sys_notice	수정	여러 공지사항 일괄 논리 삭제
sys_log_change_his	생성	삭제 이력 일괄 기록

Error:

- 권한 없음
- 일부 공지사항 없음 (트랜잭션 롤백)
- 트랜잭션 실패

관련 API: `POST /api/admin/notices/list-delete` (상세는 API 규격서 참조)

5.4.4 콘텐츠 관리 공통 특징

Paranoid Delete (논리 삭제):

- 물리 삭제 하지 않음
- `del_yn='Y'` 설정
- `deleted_at`, `deleted_by` 기록
- 감사 목적 (복구 가능)

변경 이력 (sys_log_change_his):

- 모든 생성/수정/삭제 기록
- JSONB 형식: `{"bf": {...}, "af": {...}}`

- 작업자 정보: `actor_type` , `actor_id`

권한 구분:

- 설계: EDITOR 이상만 생성/수정/삭제
- 권장: Controller에 `hasContentEditPermission()` 추가

5.5 공통 코드 관리 (S-ADMIN 전용)

> **권한 구분**:
>
> - **모든 API**: S-ADMIN만 접근 가능
> - **현재 구현**: `checkSuperRole()` 체크

5.5.1 코드 그룹 관리

(1) 코드 그룹 목록 조회

기능: S-ADMIN이 전체 코드 그룹 목록을 조회하고 그룹별 코드 개수를 확인하는 기능

권한: S-ADMIN만

처리 Flow:

```
1. 권한 체크 (checkSuperRole)
2. 쿼리 파라미터 파싱
3. DB 조회 (sys_common_code, GROUP BY grp_id)
4. 그룹별 코드 개수 계산
5. 응답 반환
```

DB 테이블 참조:

테이블	용도	설명
`sys_common_code`	조회	코드 그룹 목록 및 그룹별 통계

쿼리 파라미터:

- `search` : 검색어 (그룹 ID, 그룹명)
- `useYn` : 사용 여부 필터 (Y/N)

Error:

- S-ADMIN 권한 없음

관련 API: `GET /api/common-code/groups` (상세는 API 규격서 참조)

(2) 코드 그룹 생성

기능: 새 코드 그룹을 생성하고 그룹에 속한 코드들을 한 번에 등록하는 기능

권한: S-ADMIN만

처리 Flow:

1. 권한 체크 (checkSuperRole)
2. 요청 검증
3. 그룹 ID 중복 확인
4. DB 트랜잭션 시작
5. 그룹에 속한 코드들 일괄 생성
6. 트랜잭션 커밋
7. 변경 로그 기록
8. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_common_code	조회	그룹 ID 중복 확인
sys_common_code	생성	그룹 + 코드들 일괄 생성 (트랜잭션)
sys_log_change_his	생성	코드 그룹 생성 이력 기록

Error:

- S-ADMIN 권한 없음
- 그룹 ID 중복

- 필수 필드 누락
- 트랜잭션 실패

주요 특징: 트랜잭션으로 그룹 + 코드들 원자적 생성 (전체 성공 또는 전체 실패)

관련 API: `POST /api/common-code/groups` (상세는 API 규격서 참조)

(3) 코드 그룹 수정

기능: 코드 그룹의 이름과 설명을 수정하는 기능

권한: S-ADMIN만

처리 Flow:

1. 권한 체크
2. 그룹 존재 확인
3. 요청 검증
4. DB 업데이트 (sys_common_code WHERE grp_id = ?)
5. 변경 로그 기록
6. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_common_code	조회	수정 대상 그룹 존재 확인
sys_common_code	수정	그룹명(grp_nm) 업데이트
sys_log_change_his	생성	변경 이력 기록

Error:

- S-ADMIN 권한 없음
- 그룹 없음

주의: 그룹 ID는 변경 불가 (시스템 전체 영향)

관련 API: `PUT /api/common-code/groups/:grpId` (상세는 API 규격서 참조)

(4) 코드 그룹 삭제

기능: 코드 그룹과 그룹에 속한 모든 코드를 논리적으로 삭제하는 기능

권한: S-ADMIN만

처리 Flow:

1. 권한 체크
2. 그룹 존재 확인
3. 시스템 코드 그룹 보호 체크 (sys_admin_roles 등)
4. 트랜잭션: 그룹 내 모든 코드 논리 삭제
5. 변경 로그 기록
6. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_common_code	조회	삭제 대상 그룹 존재 확인
sys_common_code	수정	그룹 내 모든 코드 논리 삭제
sys_log_change_his	생성	삭제 이력 기록

Error:

- S-ADMIN 권한 없음
- 그룹 없음
- **시스템 코드 그룹 삭제 시도**

보안: `sys_admin_roles`, `sys_data_status` 등 시스템 필수 코드 그룹은 삭제 불가

관련 API: `DELETE /api/common-code/groups/:grpId` (상세는 API 규격서 참조)

5.5.2 개별 코드 관리

(1) 코드 추가

기능: 기존 코드 그룹에 새로운 코드를 추가하는 기능

권한: S-ADMIN만

처리 Flow:

1. 권한 체크
2. 그룹 존재 확인
3. 코드 ID 중복 확인
4. DB 생성 (sys_common_code)
5. 변경 로그 기록
6. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_common_code	조회	그룹 존재 및 코드 중복 확인
sys_common_code	생성	새 코드 추가
sys_log_change_his	생성	코드 생성 이력 기록

Error:

- S-ADMIN 권한 없음
- 그룹 없음
- 코드 ID 중복

관련 API: `POST /api/common-code/codes/:grpId` (상세는 API 규격서 참조)

(2) 코드 수정

기능: 코드의 이름, 설명, 정렬 순서를 수정하는 기능

권한: S-ADMIN만

처리 Flow:

1. 권한 체크
2. 코드 존재 확인
3. 요청 검증
4. DB 업데이트 (sys_common_code)
5. 변경 로그 기록
6. 응답 반환

DB 테이블 참조:

테이블	용도	설명
sys_common_code	조회	수정 대상 코드 존재 확인
sys_common_code	수정	코드명, 설명, 정렬 순서 업데이트
sys_log_change_his	생성	변경 이력 기록

Error:

- S-ADMIN 권한 없음
- 코드 없음

주의: 코드 ID는 변경 불가 (시스템 전체 영향, Frontend에서 참조)

관련 API: `PUT /api/common-code/codes/:grpId/:codeId` (상세는 API 규격서 참조)

(3) 코드 삭제

기능: 개별 코드를 논리적으로 삭제하는 기능

권한: S-ADMIN만

처리 Flow:

1. 권한 체크
2. 코드 존재 확인
3. 시스템 필수 코드 보호 체크
4. 논리 삭제 (del_yn='Y')
5. 변경 로그 기록
6. 응답 반환

DB 테이블 참조:

테이블	용도	설명
`sys_common_code`	조회	삭제 대상 코드 존재 확인
`sys_common_code`	수정	논리 삭제 처리
`sys_log_change_his`	생성	삭제 이력 기록

Error:

- S-ADMIN 권한 없음
- 코드 없음
- **시스템 필수 코드 삭제 시도**

보안: S-ADMIN, ADMIN, EDITOR, VIEWER 등 역할 코드는 삭제 불가

관련 API: `DELETE /api/common-code/codes/:grpId/:codeId` (상세는 API 규격서 참조)

5.5.3 공통 코드 조회 (Public)

(1) 그룹별 코드 조회

기능: Frontend에서 드롭다운, 필터 등에 사용할 코드 목록을 조회하는 기능

권한: Public

처리 Flow:

```
1. DB 조회 (sys_common_code WHERE grp_id = ?, use_yn='Y')
2. 응답 반환
```

DB 테이블 참조:

테이블	용도	설명
`sys_common_code`	조회	그룹별 코드 목록 (사용 중인 것만)

Error: 없음 (빈 배열 반환 가능)

활용 예시:

- FAQ 타입 드롭다운
- 공지사항 유형 필터
- 상태 선택 옵션

관련 API: `GET /api/common-code/:grpId` (상세는 API 규격서 참조)

(2) 코드 상세 조회

기능: 특정 코드의 상세 정보를 조회하는 기능

권한: Public

DB 테이블 참조:

테이블	용도	설명
`sys_common_code`	조회	코드 상세 정보 조회

Error:

- 코드 없음

관련 API: `GET /api/common-code/:grpId/:codeId` (상세는 API 규격서 참조)

5.5.4 공통 코드 시스템 특징

시스템 코드 보호:

- `sys_admin_roles` : 관리자 역할 (S-ADMIN, ADMIN, EDITOR, VIEWER)
- `sys_data_status` : 데이터 상태
- `sys_work_type` : 작업 유형

→ 이러한 시스템 필수 코드는 삭제/수정 시 검증 로직 필요

계층 구조 지원:

- `parent_grp_id` , `parent_code_id` : 부모 코드 참조
- `code_lvl` : 코드 레벨 (1, 2, 3...)

- `sort_order` : 정렬 순서

코드 타입:

- `B` (Basic): 기본 코드
- `A` (Advanced): 고급 코드
- `S` (System): 시스템 코드

활용:

- Frontend: 드롭다운, 필터, 라벨 표시
- Backend: 상태 검증, 타입 체크

5.6 OpenAPI 키 관리

> **핵심 비즈니스 기능**: IITP DABT 플랫폼의 Open API 서비스 제공을 위한 인증키 관리
>
> **권한 구분**:
>
> - **사용자 (User)**: 본인 키만 조회/신청/연장/삭제
> - **관리자 (Admin)**: 전체 키 조회/승인/거부/관리
> - **현재 구현**: `authMiddleware` (User), `adminAuthMiddleware` (Admin) ✅

5.6.1 사용자 OpenAPI 키 관리 (User)

(1) 본인 API 키 목록 조회

기능: 사용자가 본인이 신청한 OpenAPI 인증키 목록을 조회하는 기능

권한: 사용자 인증 필요

처리 Flow:

1. 권한 체크 (authMiddleware)
2. req.user.userId 추출
3. DB 조회 (open_api_auth_key WHERE user_id = ?)
4. 응답 반환

DB 테이블 참조:

테이블	용도	설명
open_api_auth_key	조회	본인의 API 키 목록 조회

Error:

- 인증 실패 → ErrorCode.UNAUTHORIZED (401)

관련 API: GET /api/user/open-api (상세는 API 규격서 참조)

(2) API 키 상세 조회

기능: 특정 API 키의 상세 정보를 조회하는 기능 (본인 키만)

권한: 사용자 인증 필요

처리 Flow:

1. 권한 체크
2. DB 조회 (키 존재 및 소유권 확인)
3. 응답 반환

DB 테이블 참조:

테이블	용도	설명
open_api_auth_key	조회	API 키 상세 정보 및 소유권 확인

Error:

- 키 없음 → ErrorCode.API_KEY_NOT_FOUND (404)
- 다른 사용자의 키 → ErrorCode.FORBIDDEN (403)

관련 API: `GET /api/user/open-api/:keyId` (상세는 API 규격서 참조)

(3) API 키 신청

기능: 사용자가 새로운 OpenAPI 인증키를 신청하는 기능

권한: 사용자 인증 필요

처리 Flow:

```
1. 권한 체크
2. 요청 검증 (키 이름, 설명)
3. API 키 생성 (utils/authKeyGenerator.ts)
   - crypto.randomBytes(30) 기반 60자 16진수 문자열 생성
   - 형식: /^[a-f0-9]{60}$/
4. DB 생성 (open_api_auth_key)
   - active_yn='N' (관리자 승인 대기)
   - user_id 설정
5. 응답 반환 (키 ID)
```

DB 테이블 참조:

테이블	용도	설명
`open_api_auth_key`	생성	새 API 키 신청 (승인 대기 상태)

Error:

- 키 이름 검증 실패 → `ErrorCode.VALIDATION_ERROR` (400)
- 키 설명 검증 실패 → `ErrorCode.VALIDATION_ERROR` (400)

주요 특징:

- 신청 즉시 생성되지만 `active_yn='N'` (비활성)
- 관리자 승인 후 활성화 (`active_yn='Y'` , `active_at` 기록)
- crypto.randomBytes 기반 고유 키로 중복 불가
- 60자 16진수 문자열 형식 (hex)

관련 API: `POST /api/user/open-api` (상세는 API 규격서 참조)

(4) API 키 연장 요청

기능: 기존 API 키의 유효기간 연장을 요청하는 기능

권한: 사용자 인증 필요

처리 Flow:

1. 권한 체크
2. 키 존재 및 소유권 확인
3. 활성화 상태 확인
4. 연장 요청 기록
5. DB 업데이트
6. 응답 반환

DB 테이블 참조:

테이블	용도	설명
`open_api_auth_key`	조회	연장 대상 키 확인
`open_api_auth_key`	수정	연장 요청 기록

Error:

- 키 없음 → `ErrorCode.API_KEY_NOT_FOUND` (404)
- 소유권 없음 → `ErrorCode.FORBIDDEN` (403)
- 비활성 키 → `ErrorCode.API_KEY_INACTIVE` (403)

관련 API: `POST /api/user/open-api/extend` (상세는 API 규격서 참조)

(5) API 키 삭제

기능: 사용자가 본인의 API 키를 논리 삭제하는 기능

권한: 사용자 인증 필요

처리 Flow:

1. 권한 체크
2. 키 존재 및 소유권 확인
3. 논리 삭제 (del_yn='Y')
4. 변경 로그 기록
5. 응답 반환

DB 테이블 참조:

테이블	용도	설명
open_api_auth_key	조회	삭제 대상 키 확인
open_api_auth_key	수정	논리 삭제 (del_yn='Y')
sys_log_change_his	생성	삭제 이력 기록

Error:

- 키 없음 → `ErrorCode.API_KEY_NOT_FOUND` (404)
- 소유권 없음 → `ErrorCode.FORBIDDEN` (403)

관련 API: `DELETE /api/user/open-api/:keyId` (상세는 API 규격서 참조)

5.6.2 관리자 OpenAPI 키 관리 (Admin)

(1) 전체 API 키 목록 조회

기능: 관리자가 모든 사용자의 OpenAPI 키 목록을 조회하고 관리하는 기능

권한: 모든 관리자 (VIEWER 포함)

처리 Flow:

1. 권한 체크 (adminAuthMiddleware)
2. 쿼리 파라미터 파싱 (검색, 필터, 페이징)
3. DB 조회 (open_api_auth_key JOIN open_api_user)
4. 응답 반환

DB 테이블 참조:

테이블	용도	설명
`open_api_auth_key`	조회	전체 API 키 목록 조회
`open_api_user`	조회	JOIN - 사용자 정보 포함

쿼리 파라미터:

- `page` : 페이지 번호
- `limit` : 페이지당 개수
- `search` : 검색어 (키 이름, 사용자 이름/이메일)
- `activeYn` : 활성화 상태 필터 (Y/N)
- `status` : 승인 상태 필터

Error:

- 관리자 권한 없음 → `ErrorCode.FORBIDDEN` (403)

관련 API: `GET /api/admin/open-api` (상세는 API 규격서 참조)

(2) API 키 통계 조회

기능: OpenAPI 키 현황 통계를 조회하는 기능 (대시보드용)

권한: 모든 관리자

처리 Flow:

```
1. 권한 체크
2. DB 집계 쿼리
   - COUNT(*) WHERE del_yn='N' (전체)
   - COUNT(*) WHERE active_yn='Y' (활성)
   - COUNT(*) WHERE active_yn='N' AND del_yn='N' (승인 대기)
3. 응답 반환
```

DB 테이블 참조:

테이블	용도	설명
`open_api_auth_key`	조회	통계 집계 (COUNT, GROUP BY)

응답 데이터:

- totalKeys: 전체 키 수
- activeKeys: 활성 키 수
- inactiveKeys: 비활성 키 수
- pendingKeys: 승인 대기 키 수

관련 API: `GET /api/admin/open-api/status` (상세는 API 규격서 참조)

(3) API 키 상세 조회

기능: 특정 사용자의 API 키 상세 정보를 조회하는 기능

권한: 모든 관리자

처리 Flow:

```
1. 권한 체크
2. DB 조회 (open_api_auth_key JOIN open_api_user)
3. 응답 반환
```

DB 테이블 참조:

테이블	용도	설명
`open_api_auth_key`	조회	API 키 상세 정보
`open_api_user`	조회	사용자 정보 (JOIN)

Error:

- 키 없음 → `ErrorCode.API_KEY_NOT_FOUND` (404)

관련 API: `GET /api/admin/open-api/:keyId` (상세는 API 규격서 참조)

(4) API 키 직접 생성

기능: 관리자가 특정 사용자를 위한 API 키를 직접 생성하는 기능 (즉시 활성화)

권한: ADMIN 이상

처리 Flow:

```
1. 권한 체크
2. 대상 사용자 존재 확인 (user_id)
3. 요청 검증 (키 이름, 설명, 유효기간)
4. API 키 생성 (authKeyGenerator.ts)
   - crypto.randomBytes(30) 기반 60자 16진수 문자열 생성
   - 형식: /^[a-f0-9]{60}$/
5. DB 생성 (open_api_auth_key)
   - active_yn='Y' (즉시 활성화)
   - start_dt, end_dt 설정
   - active_at 기록
6. 변경 로그 기록
7. 응답 반환
```

DB 테이블 참조:

테이블	용도	설명
`open_api_user`	조회	대상 사용자 존재 확인
`open_api_auth_key`	생성	새 API 키 생성 (즉시 활성)
`sys_log_change_his`	생성	키 생성 이력 기록

Error:

- 권한 없음 → `ErrorCode.FORBIDDEN` (403)
- 사용자 없음 → `ErrorCode.USER_NOT_FOUND` (404)
- 키 이름 중복 → `ErrorCode.VALIDATION_ERROR` (400)

사용자 신청과의 차이:

- 사용자 신청: `active_yn='N'` (승인 대기)
- 관리자 생성: `active_yn='Y'` (즉시 활성)

관련 API: `POST /api/admin/open-api` (상세는 API 규격서 참조)

(5) API 키 수정 (승인/거부)

기능: 사용자가 신청한 API 키를 승인하거나 거부하는 기능

권한: ADMIN 이상

처리 Flow:

```
1. 권한 체크
2. 키 존재 확인
3. 승인 시:
   - active_yn='Y' 설정
   - active_at 기록
   - start_dt, end_dt 설정
4. 거부 시:
   - active_yn='N' 유지
   - key_reject_reason 기록
5. DB 업데이트
6. 변경 로그 기록
7. 응답 반환
```

DB 테이블 참조:

테이블	용도	설명
`open_api_auth_key`	조회	대상 키 확인
`open_api_auth_key`	수정	승인 상태 및 사유 업데이트
`sys_log_change_his`	생성	승인/거부 이력 기록

Error:

- 권한 없음 → `ErrorCode.FORBIDDEN` (403)
- 키 없음 → `ErrorCode.API_KEY_NOT_FOUND` (404)
- 이미 활성화된 키 → `ErrorCode.API_KEY_ALREADY_ACTIVE` (400)

관련 API: `PUT /api/admin/open-api/:keyId` (상세는 API 규격서 참조)

(6) API 키 연장 승인

기능: 사용자의 API 키 연장 요청을 승인하여 유효기간을 연장하는 기능

권한: ADMIN 이상

처리 Flow:

1. 권한 체크
2. 키 존재 확인
3. 연장 기간 계산 (예: 현재 종료일 + 90일)
4. DB 업데이트 (end_dt 연장)
5. 변경 로그 기록
6. 응답 반환

DB 테이블 참조:

테이블	용도	설명
open_api_auth_key	조회	연장 대상 키 확인
open_api_auth_key	수정	유효기간(end_dt) 연장
sys_log_change_his	생성	연장 이력 기록

Error:

- 권한 없음 → `ErrorCode.FORBIDDEN` (403)
- 키 없음 → `ErrorCode.API_KEY_NOT_FOUND` (404)
- 비활성 키 → `ErrorCode.API_KEY_INACTIVE` (403)

관련 API: `POST /api/admin/open-api/:keyId/extend` (상세는 API 규격서 참조)

(7) API 키 삭제

기능: 특정 사용자의 API 키를 논리 삭제하는 기능

권한: ADMIN 이상

처리 Flow:

1. 권한 체크
2. 키 존재 확인
3. 논리 삭제 (del_yn='Y')
4. 변경 로그 기록
5. 응답 반환

DB 테이블 참조:

테이블	용도	설명
open_api_auth_key	조회	삭제 대상 키 확인
open_api_auth_key	수정	논리 삭제 (del_yn='Y')
sys_log_change_his	생성	삭제 이력 기록

Error:

- 권한 없음 → `ErrorCode.FORBIDDEN` (403)
- 키 없음 → `ErrorCode.API_KEY_NOT_FOUND` (404)

관련 API: `DELETE /api/admin/open-api/:keyId` (상세는 API 규격서 참조)

(8) API 키 일괄 삭제

기능: 선택한 여러 API 키를 한 번에 논리 삭제하는 기능

권한: ADMIN 이상

처리 Flow:

1. 권한 체크
2. 키 ID 배열 검증 (keyIds: number[])
3. 트랜잭션 시작
4. 각 키 존재 확인
5. 일괄 논리 삭제
6. 변경 로그 일괄 기록
7. 트랜잭션 커밋
8. 응답 반환

DB 테이블 참조:

테이블	용도	설명
`open_api_auth_key`	조회	대상 키들 존재 확인
`open_api_auth_key`	수정	여러 키 일괄 논리 삭제
`sys_log_change_his`	생성	삭제 이력 일괄 기록

Error:

- 권한 없음 → `ErrorCode.FORBIDDEN` (403)
- 일부 키 없음 → 트랜잭션 롤백
- 트랜잭션 실패 → `ErrorCode.DATABASE_ERROR` (500)

주요 특징: 트랜잭션으로 전체 성공 또는 전체 실패

관련 API: `POST /api/admin/open-api/list-delete` (상세는 API 규격서 참조)

5.6.3 OpenAPI 키 시스템 특징

키 생성 알고리즘:

- UUID v4 기반 고유 키 생성
- 중복 불가 보장
- 구현: `utils/authKeyGenerator.ts`

키 라이프사이클:

```
[신청] → [승인 대기] → [승인/거부] → [활성] → [연장] → [만료/삭제]
   ↓          ↓              ↓          ↓       ↓        ↓
User     active_yn='N'   Admin     active_yn='Y'  User   del_yn='Y'
                                   start_dt/end_dt
                                   active_at 기록
```

상태 관리:

- `active_yn` : Y(활성) / N(비활성)
- `del_yn` : Y(삭제) / N(정상)
- `start_dt` , `end_dt` : 유효 기간
- `active_at` : 활성화 시각

- `latest_acc_at` : 최종 접근 시각 (API 호출 시 업데이트)

보안:

- 사용자는 본인 키만 접근 가능
- 관리자는 전체 키 관리 가능
- 삭제된 키는 복구 가능 (논리 삭제)
- 만료된 키는 API 호출 불가

워크플로우:

1. **사용자 신청** → `active_yn='N'` (대기)
2. **관리자 승인** → `active_yn='Y'` , `active_at` 기록, 유효기간 설정
3. **사용자 사용** → `latest_acc_at` 업데이트
4. **만료 임박 시** → 사용자가 연장 요청
5. **관리자 연장 승인** → `end_dt` 연장
6. **불필요 시** → 사용자/관리자가 삭제

6. 데이터베이스 설계

> **전체 DB 개요**: 프로젝트 아키텍처 가이드 6장 참조

6.1 Sequelize 모델

6.1.1 모델 초기화 (models/index.ts)

데이터베이스 연결:

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

const dbHost = process.env.DB_HOST || 'localhost';
const dbPort = parseInt(process.env.DB_PORT || '5432');
const dbName = process.env.DB_NAME || 'iitp_dabt_admin';
const dbUser = process.env.DB_USER || 'postgres';
const dbPassword = getDecryptedEnv('DB_PASSWORD') || '';

// Sequelize 인스턴스 생성
const sequelize = new Sequelize(dbName, dbUser, dbPassword, {
  host: dbHost,
  port: dbPort,
  dialect: 'postgres',
  logging: process.env.NODE_ENV === 'development' ? appLogger.info : false,
  pool: {
    max: 5,
    min: 0,
    acquire: 30000,
    idle: 10000
  }
});

// 모델 초기화
initOpenApiUser(sequelize);
initOpenApiAuthKey(sequelize);
initSysAdmAccount(sequelize);
initSysCommonCode(sequelize);
initSysFaq(sequelize);
initSysQna(sequelize);
initSysNotice(sequelize);
initSysLogUserAccess(sequelize);
initSysLogChangeHis(sequelize);

export default sequelize;
```

6.1.2 모델 파일 구조

모델 파일	테이블명	용도
openApiUser.ts	open_api_user	일반 사용자 계정

모델 파일	테이블명	용도
openApiAuthKey.ts	open_api_auth_key	OpenAPI 인증 키
sysAdmAccount.ts	sys_adm_account	관리자 계정
sysCommonCode.ts	sys_common_code	공통 코드
sysFaq.ts	sys_faq	FAQ
sysQna.ts	sys_qna	Q&A
sysNotice.ts	sys_notice	공지사항
sysLogUserAccess.ts	sys_log_user_access	접근 로그
sysLogChangeHis.ts	sys_log_change_his	변경 로그

6.2 주요 테이블 상세

6.2.1 open_api_user (일반 사용자)

테이블명: `open_api_user`

용도: 일반 사용자 계정 정보

스키마:

- **PK**: `user_id` (INTEGER, AUTO_INCREMENT)
- **UK**: `login_id` (이메일)

주요 컬럼:

- `login_id` : 로그인 이메일 (VARCHAR(100), UNIQUE)
- `password` : bcrypt 해시 (CHAR(60))
- `user_name` : 사용자 이름 (VARCHAR(100))
- `status` : 계정 상태 (VARCHAR(20))
- `del_yn` : 삭제 여부 (CHAR(1), 기본 'N')
- `affiliation` : 소속 (VARCHAR(100))

- `latest_login_at` : 최근 로그인 시각 (TIMESTAMP)

인덱스:

- UNIQUE(login_id)

특징:

- 논리 삭제 지원 (`del_yn`)
- Sequelize Paranoid 미사용

6.2.2 sys_adm_account (관리자)

테이블명: `sys_adm_account`

용도: 관리자 계정 및 역할 정보

스키마:

- **PK**: `adm_id` (INTEGER, AUTO_INCREMENT)
- **UK**: `login_id` (이메일)

주요 컬럼:

- `login_id` : 관리자 이메일 (VARCHAR(100), UNIQUE)
- `password` : bcrypt 해시 (CHAR(60))
- `name` : 관리자 이름 (VARCHAR(100))
- `roles` : 역할 코드 (VARCHAR(20)) - S-ADMIN, ADMIN, EDITOR, VIEWER
- `status` : 계정 상태 (VARCHAR(20))
- `del_yn` : 삭제 여부 (CHAR(1), 기본 'N')
- `affiliation` : 소속 (VARCHAR(100))
- `description` : 설명 (VARCHAR(500))

인덱스:

- UNIQUE(login_id)
- INDEX(roles, status)

특징:

- 논리 삭제 지원

- **roles 컬럼**: 권한 체계의 핵심

6.2.3 sys_common_code (공통 코드)

테이블명: `sys_common_code`

용도: 시스템 공통 코드

스키마:

- **PK**: (`grp_id` , `code_id`) (복합키)

주요 컬럼:

- `grp_id` : 코드 그룹 ID (VARCHAR(50))
- `grp_nm` : 코드 그룹 이름 (VARCHAR(100))
- `code_id` : 코드 ID (VARCHAR(50))
- `code_nm` : 코드 이름 (VARCHAR(100))
- `code_type` : 코드 타입 (CHAR(1)) - B/A/S
- `parent_grp_id` : 부모 그룹 ID (VARCHAR(50))
- `parent_code_id` : 부모 코드 ID (VARCHAR(50))
- `code_lvl` : 코드 레벨 (INTEGER)
- `sort_order` : 정렬 순서 (INTEGER)
- `use_yn` : 사용 여부 (CHAR(1), 기본 'Y')
- `del_yn` : 삭제 여부 (CHAR(1), 기본 'N')

인덱스:

- PRIMARY KEY(grp_id, code_id)
- INDEX(grp_id, use_yn, del_yn)

특징:

- 계층 구조 지원 (parent_grp_id, parent_code_id)
- 시스템 코드 보호 로직 필요

6.2.4 sys_faq / sys_qna / sys_notice (콘텐츠)

공통 특징:

- 논리 삭제 지원 (`del_yn='Y'`)
- 생성/수정/삭제 시 `sys_log_change_his` 기록
- `created_by` , `updated_by` , `deleted_by` 컬럼으로 작업자 추적

sys_faq:

- PK: `faq_id`
- 주요 컬럼: `question` , `answer` , `use_yn`

sys_qna:

- PK: `qna_id`
- FK: `user_id` → `open_api_user.user_id`
- 주요 컬럼: `question` , `answer` , `answered_at` , `status`

sys_notice:

- PK: `notice_id`
- 주요 컬럼: `title` , `content` , `pinned_yn` , `public_yn` , `start_dt` , `end_dt`

6.2.5 sys_log_user_access (접근 로그)

테이블명: `sys_log_user_access`

용도: 로그인/로그아웃 이력 기록

스키마:

- **PK**: `log_id` (BIGINT, AUTO_INCREMENT)

주요 컬럼:

- `user_id` : 사용자/관리자 ID (INTEGER)
- `user_type` : 사용자 타입 (CHAR(1)) - U/A
- `log_type` : 로그 타입 (VARCHAR(8)) - LOGIN/LOGOUT
- `act_result` : 결과 (CHAR(1)) - S(성공)/F(실패)
- `err_code` : 에러 코드 (VARCHAR(10))

- `err_msg` : 에러 메시지 (VARCHAR(200))
- `ip_addr` : IP 주소 (VARCHAR(50))
- `user_agent` : User Agent (VARCHAR(512))
- `access_tm` : 접근 시각 (TIMESTAMP)

인덱스:

- INDEX(user_id, user_type, log_type)

특징:

- 성공/실패 모두 기록
- 보안 감사용

6.2.6 sys_log_change_his (변경 로그)

테이블명: `sys_log_change_his`

용도: 데이터 변경 이력 추적 (Audit Log)

스키마:

- **PK**: `log_id` (BIGINT, AUTO_INCREMENT)

주요 컬럼:

- `actor_type` : 작업자 타입 (CHAR(1)) - U/A
- `actor_id` : 작업자 ID (BIGINT)
- `action_type` : 액션 타입 (VARCHAR(36)) - CREATE/UPDATE/DELETE
- `target_type` : 대상 타입 (VARCHAR(64)) - USER/ADMIN/FAQ 등
- `target_id` : 대상 ID (BIGINT)
- `act_result` : 결과 (CHAR(1)) - S/F
- `chg_summary` : 변경 내용 (JSONB) - {"bf": {...}, "af": {...}}
- `err_code` : 에러 코드 (VARCHAR(10))
- `err_msg` : 에러 메시지 (VARCHAR(200))
- `ip_addr` : IP 주소 (VARCHAR(50))
- `act_tm` : 액션 시각 (TIMESTAMP)

인덱스:

- INDEX(actor_type, actor_id, action_type)

- INDEX(target_type, target_id)

특징:

- **JSONB 컬럼**: 변경 전후 데이터 저장
- 모든 중요 작업 기록
- 감사 및 복구 목적

변경 내용 예시:

```
{
  "bf": { "name": "홍길동", "affiliation": "A팀" },
  "af": { "name": "홍길동", "affiliation": "B팀" }
}
```

6.2.7 open_api_auth_key (OpenAPI 인증키)

테이블명: `open_api_auth_key`

용도: OpenAPI 인증키 발급 및 관리

스키마:

- **PK**: `key_id` (INTEGER, AUTO_INCREMENT)
- **FK**: `user_id` → `open_api_user.user_id`
- **UK**: `auth_key` (인증 키 문자열, UNIQUE)

주요 컬럼:

- `key_id` : 키 고유 ID (INTEGER)
- `user_id` : 소유자 사용자 ID (INTEGER, FK, NOT NULL)
- `auth_key` : 인증 키 문자열 (VARCHAR(255), UNIQUE, NOT NULL)
- `active_yn` : 활성화 여부 (CHAR(1), DEFAULT 'N') - Y/N
- `start_dt` : 유효 시작일 (DATE, NULLABLE)
- `end_dt` : 유효 종료일 (DATE, NULLABLE)
- `del_yn` : 삭제 여부 (CHAR(1), DEFAULT 'N')
- `key_name` : 키 이름 (VARCHAR(100), NOT NULL)
- `key_desc` : 키 설명 (TEXT, NULLABLE)
- `key_reject_reason` : 거부 사유 (TEXT, NULLABLE) - 거부 시에만

- `active_at` : 활성화 시각 (TIMESTAMP, NULLABLE)
- `latest_acc_at` : 최종 접근 시각 (TIMESTAMP, NULLABLE)
- `created_at` , `updated_at` , `deleted_at` : Sequelize timestamps
- `created_by` , `updated_by` , `deleted_by` : 작업자 추적 (VARCHAR)

인덱스:

- UNIQUE(auth_key) - 키 문자열 중복 방지
- INDEX(user_id, active_yn) - 사용자별 활성 키 조회
- INDEX(active_yn, del_yn) - 전체 활성/대기 키 조회
- INDEX(end_dt) - 만료 키 조회

특징:

- **논리 삭제 지원**: `del_yn='Y'` (복구 가능)
- **유효기간 관리**: `start_dt` , `end_dt` 로 기간 제한
- **승인 워크플로우**: `active_yn='N'` (대기) → `active_yn='Y'` (승인)
- **거부 처리**: `key_reject_reason` 에 거부 사유 기록
- **사용 추적**: `latest_acc_at` 으로 마지막 API 호출 시각 기록

관계:

- **belongsTo**: `open_api_user` (user_id → user_id)
  - 한 사용자는 여러 API 키 보유 가능
  - CASCADE 옵션 없음 (사용자 삭제 시 키 유지)

모델 파일: `src/models/openApiAuthKey.ts`

활용:

- Frontend: 사용자 "My API Keys" 화면, 관리자 "API 키 관리" 화면
- Backend: API 인증 (Auth Key 검증)
- External API: 외부 시스템이 이 키로 IITP DABT Platform API 호출

보안:

- `auth_key` 는 UUID v4 (36자, 예: `550e8400-e29b-41d4-a716-446655440000`)
- 추측 불가능한 고유값
- HTTPS 통신 필수

7. 환경 설정 및 배포

7.1 환경 변수

> **전체 환경 변수 목록**: 프로젝트 아키텍처 가이드 Appendix D 참조

7.1.1 필수 환경 변수 (.env)

```
# 서버 설정
NODE_ENV=production
PORT=30000

# 데이터베이스
DB_HOST=your-db-host
DB_PORT=5432
DB_NAME=iitp_dabt_admin
DB_USER=your-db-user
DB_PASSWORD=your-db-password  # 또는 ENC(암호화된값)

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

# 암호화
ENC_SECRET=your-encryption-secret

# CORS
CORS_ORIGINS=https://your-domain.com,https://www.your-domain.com

# 로깅
LOG_LEVEL=warn
```

7.2 빌드 및 배포 (간략)

> **상세 가이드**: 서버 배포 및 설치 가이드 참조

7.2.1 로컬 개발 빌드

```
# 1. 의존성 설치
npm install

# 2. Common 패키지 빌드 (필수)
cd ../packages/common && npm run build && cd ../../be

# 3. Backend 빌드
npm run build

# 4. 실행
npm start
```

빌드 결과물: `be/dist/`

7.2.2 서버 배포 (PM2)

```
# PM2로 실행
pm2 start dist/index.js --name iitp-dabt-adm-be

# 재시작
pm2 restart iitp-dabt-adm-be

# 로그 확인
pm2 logs iitp-dabt-adm-be

# 상태 확인
pm2 status
```

PM2 설정: `script/start-server-be.js` 참조

7.2.3 배포 전 체크리스트

- [ ] `.env` 파일 설정 완료 (운영 환경 값)
- [ ] `NODE_ENV=production` 설정
- [ ] DB 연결 정보 확인
- [ ] JWT_SECRET 강력한 값으로 설정
- [ ] 민감 정보 암호화 (ENC(...))
- [ ] CORS_ORIGINS 정확히 설정
- [ ] LOG_LEVEL=warn 설정 (운영)
- [ ] Common 패키지 빌드 완료

상세 배포 절차: 서버 배포 및 설치 가이드 참조

7.3 로깅 (Winston 3-File Strategy)

> **상세 설명**: 프로젝트 아키텍처 가이드 섹션 8.3 참조

본 시스템은 **3개의 로그 파일**로 로그를 분리하여 관리합니다.

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

용도: 비즈니스 로직, 애플리케이션 이벤트

경로: `be/logs/app-YYYY-MM-DD.log`

로그 레벨: `info` 이상 (info, warn, error)

로그 형식:

```
[2024-11-06 10:30:45] [INFO] 회원가입 성공: userId=123
[2024-11-06 10:31:20] [WARN] 토큰 만료 2분 전
[2024-11-06 10:32:10] [ERROR] DB 연결 실패: connection timeout
```

사용: `appLogger.info()`, `appLogger.warn()`, `appLogger.error()`

구현 위치: `src/utils/logger.ts`

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

용도: 모든 API 요청/응답 자동 기록

경로: `be/logs/access-YYYY-MM-DD.log`

로그 레벨: `info`

로그 형식:

```
[2024-11-06 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
```

자동 기록: `accessLogMiddleware` 가 모든 API 호출 시 자동으로 기록

활용:

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

구현 위치: `src/middleware/accessLogMiddleware.ts`

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

용도: 에러만 별도 저장 (빠른 에러 추적)

경로: `be/logs/error-YYYY-MM-DD.log`

로그 레벨: `error`

로그 형식:

```
[2024-11-06 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
```

특징: 스택 트레이스 포함

7.3.4 로그 공통 설정

로그 로테이션:

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

로그 레벨 설정 (환경 변수):

```
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

7.3.5 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:

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

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

8. 보안 및 암호화

> **참고**: 보안 전반은 프로젝트 아키텍처 가이드 섹션 8 참조

8.1 환경 변수 암호화 (AES-256-CBC)

기능: 민감 정보(DB 비밀번호, JWT 비밀키) 보호

암호화 방식: AES-256-CBC (Jasypt 스타일)

8.1.1 암호화 스크립트

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

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

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

스크립트 위치: `be/scripts/encrypt-env.js`

8.1.2 암호화된 값 사용

.env 파일:

```
# 민감 정보를 암호화하여 저장
DB_PASSWORD=ENC(aGVsbG93b3JsZA==...)
JWT_SECRET=ENC(bXlzZWNyZXRrZXk=...)

# ENC_SECRET은 평문으로 (별도 관리)
ENC_SECRET=your-encryption-secret
```

8.1.3 자동 복호화 (decrypt.ts)

개념: `getDecryptedEnv()` 함수가 자동으로 감지하여 복호화

동작:

1. 환경 변수 값이 `ENC(...)` 로 시작하면 → AES-256-CBC 복호화
2. 그렇지 않으면 → 평문 그대로 반환

사용 예시:

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

// 자동 복호화
const dbPassword = getDecryptedEnv('DB_PASSWORD');
// ENC(...)이면 복호화, 아니면 평문 반환

// 데이터베이스 연결
const sequelize = new Sequelize(dbName, dbUser, dbPassword, {...});
```

구현 위치: `be/src/utils/decrypt.ts`

8.1.4 보안 주의사항

** 암호화 키 관리**:

- `ENC_SECRET` 은 **.env 파일에 평문**으로 저장
- 운영 서버에서는 **환경 변수로만** 설정 권장
- Git에 커밋하지 말 것
- 서버 접근 권한이 있는 관리자만 알아야 함

암호화 대상:

- DB_PASSWORD
- JWT_SECRET
- 기타 민감 정보

암호화 불필요:

- DB_HOST, DB_PORT (공개 정보)
- NODE_ENV, PORT (민감하지 않음)

8.2 비밀번호 해싱 (bcrypt)

기능: 사용자 비밀번호를 안전하게 저장

알고리즘: bcrypt (salt rounds: 10)

8.2.1 비밀번호 해싱

회원가입/비밀번호 변경 시:

```
import bcrypt from 'bcrypt';

// 평문 비밀번호 → 해시
const hashedPassword = await bcrypt.hash(password, 10);
// 결과: $2b$10$N9qo8uLOickgx2ZMRZoMyeIjZAgcfl...
```

저장 형식: CHAR(60) (bcrypt 해시 고정 길이)

구현 위치: Service 계층 (회원가입, 비밀번호 변경 로직)

8.2.2 비밀번호 검증

로그인 시:

```
// 평문 비밀번호와 해시 비교
const isValid = await bcrypt.compare(plainPassword, hashedPassword);
// true/false 반환
```

주의: 해시를 평문으로 되돌릴 수 없음 (단방향)

구현 위치: Service 계층 (로그인 로직)

8.2.3 비밀번호 해싱 테스트

```
# 테스트 스크립트
node scripts/test-password-hash.js "your-password"

# 결과
Plain: your-password
Hash: $2b$10$N9qo8uLOickgx2ZMRZoMyeIjZAgcfl...
Verify: true
```

스크립트 위치: `be/scripts/test-password-hash.js`

부록

Appendix A: API 응답 구조

> **참고**: Common 패키지에 정의된 표준 응답 구조 사용

A.1 성공 응답

```
{
  "result": "ok",
  "data": {
    "userId": 123,
    "name": "홍길동",
    "email": "user@example.com"
  },
  "message": "Success"
}
```

필드 설명:

- `result` : 항상 "ok"
- `data` : 실제 응답 데이터 (타입은 API마다 다름)
- `message` : 선택적 메시지

A.2 에러 응답

```
{
  "result": "error",
  "errorCode": 14000,
  "message": "이메일 또는 비밀번호가 올바르지 않습니다."
}
```

필드 설명:

- `result` : 항상 "error"

- `errorCode` : Common 패키지의 에러 코드 (11xxx~22xxx)
- `message` : 사용자에게 표시할 에러 메시지

Appendix B: 트러블슈팅

B.1 데이터베이스 연결 실패

증상: Backend 실행 시 DB 연결 에러

확인 방법:

```
# PostgreSQL 상태 확인
sudo systemctl status postgresql

# 연결 테스트
psql -h $DB_HOST -U $DB_USER -d $DB_NAME

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

원인 및 해결:

1. PostgreSQL 미실행 → `sudo systemctl start postgresql`
2. DB 비밀번호 오류 → `.env` 파일 확인
3. DB 호스트 오류 → 네트워크 확인
4. 암호화된 비밀번호 복호화 실패 → `ENC_SECRET` 확인

B.2 포트 충돌

증상: `Error: listen EADDRINUSE: address already in use :::30000`

확인 방법:

```
# 포트 사용 확인
netstat -tulpn | grep :30000
# 또는
lsof -i :30000

# 프로세스 종료
kill -9 <PID>
```

해결:

- 기존 프로세스 종료
- 또는 `.env` 에서 `PORT` 변경

B.3 JWT 토큰 에러

증상: `TOKEN_INVALID` 에러 빈번

원인:

1. JWT_SECRET 불일치 (BE 재시작 후 변경됨)
2. 시간 동기화 문제 (서버 시간 불일치)

해결:

```
# JWT_SECRET 고정 (운영 환경)
# .env 파일에 ENC(...)로 암호화하여 저장

# 시간 동기화 확인
date
```

B.4 로그 파일 용량 초과

증상: 디스크 공간 부족

확인 방법:

```
# 로그 디렉토리 용량 확인
du -sh be/logs/

# 오래된 로그 확인
ls -lh be/logs/
```

해결:

```
# 30일 이전 로그 삭제 (자동)
# Winston Daily Rotate File이 자동 관리

# 수동 삭제 (필요 시)
find be/logs/ -name "*.log" -mtime +30 -delete
```