IITP DABT Admin

Frontend 설계서

문서 버전: 1.0.0
작성일: 2025-11-14

(주)스위트케이

문서 History

버전	일자	작성자	변경 내용
1.0.0	2025-11-14	(주)스위트케이	최초 작성

목차

1. Frontend 시스템 개요

1.1 개요

IITP DABT Admin Frontend는 React 기반의 Single Page Application(SPA)으로, 사용자와 관리자를 위한 직관적인 웹 인터페이스를 제공합니다.

주요 특징:

- **Modern Stack**: React 18, TypeScript, Vite, Material-UI
- **Component-Based Architecture**: 재사용 가능한 컴포넌트 구조
- **Monorepo 구조**: Common 패키지를 통한 BE/FE 코드 공유
- **독립 세션 관리**: User/Admin 동시 로그인 지원
- **Role-Based UI Control**: 권한별 UI 차별화 (VIEWER/EDITOR/ADMIN/S-ADMIN)

1.2 Frontend 역할 및 책임

영역	책임
UI/UX	사용자 인터페이스 제공 및 사용자 경험 최적화
Client-Side 검증	입력 데이터 형식 검증 (Common 패키지 활용)
인증 상태 관리	JWT 토큰 관리, 자동 갱신, 만료 체크
권한 기반 UI 제어	역할별 메뉴/버튼 표시 제어 (UX 목적)
API 통신	Backend API 호출 및 응답 처리
에러 핸들링	사용자 친화적 에러 메시지 표시
라우팅	SPA 라우팅 및 권한 기반 접근 제어

1.3 기술 스택

1.3.1 개발 환경 (필수)

환경	버전	용도
Node.js	22.x 이상	개발 서버 실행 및 프로덕션 빌드 (필수)
npm	9.x 이상	패키지 관리

> **중요**:
>
> - **개발/빌드 시**: Node.js 필수
> - **실행(런타임) 시**: 브라우저에서 실행 (Node.js 불필요)
> - 프로젝트 시작 전 Node.js와 npm 버전을 먼저 확인하세요.

1.3.2 Core 라이브러리

패키지	버전	용도
react	^18.2.0	UI 라이브러리
react-dom	^18.2.0	React DOM 렌더링
react-router-dom	^6.20.1	SPA 라우팅
typescript	^5.x	정적 타입 체킹

1.3.3 UI 프레임워크

패키지	버전	용도
@mui/material	^5.15.0	Material-UI 컴포넌트
@mui/icons-material	^5.15.0	Material-UI 아이콘
@emotion/react	^11.11.1	CSS-in-JS (MUI 의존성)
@emotion/styled	^11.11.0	Styled Components (MUI 의존성)

1.3.4 HTTP 클라이언트 및 상태 관리

패키지	버전	용도
axios	^1.11.0	HTTP 클라이언트 (실제로는 fetch 사용)
jwt-decode	^4.0.0	JWT 토큰 디코딩

> **참고**: 실제 구현에서는 `axios` 대신 `fetch` API를 직접 사용합니다.

1.3.5 빌드 도구

패키지	버전	용도
vite	^5.0.8	빌드 도구 및 개발 서버
@vitejs/plugin-react	^4.2.1	Vite React 플러그인

1.3.6 Common 패키지

패키지	위치	용도
@iitp-dabt/common	`../packages/common`	BE/FE 공유 코드

Common 패키지 제공 기능:

- **검증 함수**: `isValidEmail`, `isValidPassword`, `isValidName` 등
- **ErrorCode 체계**: 11xxx-22xxx 범위 에러 코드
- **공통 타입**: API 요청/응답 타입
- **상수**: API URL, 관리자 역할 코드 등

1.4 Common 패키지 활용

Frontend에서 Common 패키지를 다음과 같이 활용합니다:

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

활용 예시:

1. **입력 검증**:

```
  if (!isValidEmail(email)) {
    return '유효하지 않은 이메일 형식입니다.';
  }
```

2. **ErrorCode 매핑**:

```
if (data.errorCode === ErrorCode.UNAUTHORIZED) {
  return '인증이 필요합니다.';
}
```

3. **API URL**:

```
const url = `${API_BASE_URL}${FULL_API_URLS.AUTH.USER.LOGIN}`;
```

4. **관리자 역할 체크**:

```
if (adminRole === CODE_SYS_ADMIN_ROLES.SUPER_ADMIN) {
  // S-Admin 전용 기능
}
```

1.5 참고 문서

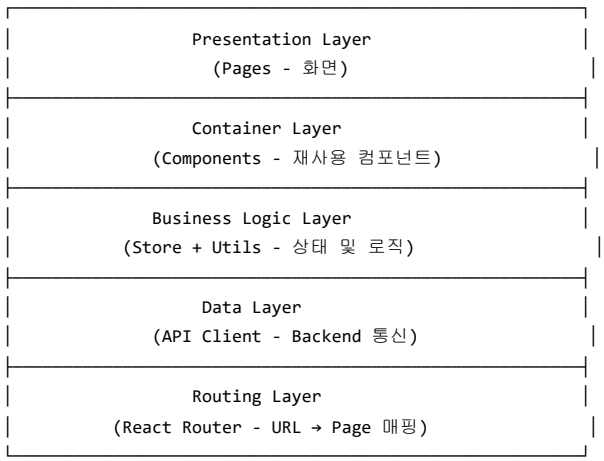
- **IITP DABT Admin 프로젝트 아키텍처 가이드** : 프로젝트 전체 아키텍처 설명
- IITP DABT Admin Backend 상세 설계서 : Backend 상세 설계서
- API 규격서 : API 스펙 상세
- 서버 배포 및 설치 가이드 : 서버 배포/설치/실행 가이드

2. Frontend 아키텍처

2.1 Frontend 아키텍처 개요 (Component-Based Architecture)

Frontend는 **Component-Based Architecture**를 기반으로 5개의 계층으로 구성됩니다.

2.1.1 계층 구조



2.1.2 계층 간 데이터 흐름

Frontend의 각 계층은 **단방향 데이터 흐름(Unidirectional Data Flow)**을 따릅니다.

요청 흐름 (Request Flow):

```
Browser (URL 입력/링크 클릭)
    ↓
React Router (Route 매칭)
    ↓
Route Guard (인증 체크: PrivateRoute/AdminProtectedRoute)
    ↓
Page Component (Lazy Loading + Suspense)
    ↓
useDataFetching Hook (상태 관리)
    ↓
API Client (apiFetch/publicApiFetch)
    ↓
Backend API (HTTP Request)
```

응답 흐름 (Response Flow):

```
Backend API (HTTP Response)
    ↓
API Client (응답 파싱, 에러 처리, 토큰 갱신)
    ↓
useDataFetching Hook (상태 업데이트: loading → success/error)
    ↓
Page Component (리렌더링 트리거)
    ↓
Container Components (Props 전달)
    ↓
Presentation Layer (UI 렌더링 완료)
    ↓
Browser (화면 표시)
```

상태 관리 흐름:

- **로딩 중**: `isLoading = true` → `LoadingSpinner` 표시
- **성공**: `data` 업데이트 → 실제 데이터 UI 렌더링
- **에러**: `isError = true` → `ErrorAlert` 표시
- **빈 데이터**: `isEmpty = true` → `EmptyState` 표시

2.2 계층 구조 상세

2.2.1 프레젠테이션 계층 (Presentation Layer) - Pages

역할: 사용자에게 보여지는 실제 화면 구성

구성:

- `/src/pages/public` - 공개 페이지 (로그인 불필요)
- `/src/pages/user` - 일반 사용자 페이지 (User 인증 필요)
- `/src/pages/admin` - 관리자 페이지 (Admin 인증 필요)

특징:

- 각 페이지는 Container 컴포넌트를 조합하여 구성
- Business Logic은 최소화하고 표현에 집중
- API 호출은 Data Layer를 통해 수행

예시:

```
// src/pages/admin/FaqList.tsx
export default function FaqList() {
  const [faqs, setFaqs] = useState([]);

  useEffect(() => {
    fetchFaqs(); // Data Layer 호출
  }, []);

  return (
    <Layout>
      <PageHeader title="FAQ 관리" />
      <DataTable data={faqs} />
    </Layout>
  );
}
```

2.2.2 컨테이너 계층 (Container Layer) - Components

역할: 재사용 가능한 UI 컴포넌트 제공

구성:

- `/src/components/common` - 범용 컴포넌트 (DataTable, Pagination 등)
- `/src/components/admin` - 관리자 전용 컴포넌트 (SideNav, AdminPageHeader)
- `/src/components` - 기타 공통 컴포넌트 (Layout, Footer 등)

특징:

- Props를 통한 데이터 전달
- 재사용성 극대화
- 비즈니스 로직 포함 최소화

예시:

```
// src/components/common/DataTable.tsx
export function DataTable({ data, columns, onRowClick }) {
  return (
    <TableContainer>
      <Table>
        {/* 테이블 렌더링 */}
      </Table>
    </TableContainer>
  );
}
```

2.2.3 비즈니스 로직 계층 (Business Logic Layer) - Store + Utils

역할: 상태 관리 및 비즈니스 로직 처리

구성:

- `/src/store/auth.ts` - 인증 상태 관리 (토큰 저장/조회/갱신)
- `/src/store/user.ts` - 사용자 정보 관리
- `/src/utils/auth.ts` - 권한 체크 함수
- `/src/utils/jwt.ts` - JWT 토큰 처리
- `/src/utils/date.ts` - 날짜 포맷팅
- `/src/utils/openApiStatus.ts` - OpenAPI 상태 처리

특징:

- LocalStorage를 통한 상태 영속화
- User/Admin 독립 세션 관리 (prefix 분리)
- 순수 함수 중심

예시:

```
// src/store/auth.ts
export function saveTokens(accessToken: string, refreshToken: string) {
  const prefix = getCurrentPrefix(); // user_ or admin_
  localStorage.setItem(prefix + 'accessToken', accessToken);
  localStorage.setItem(prefix + 'refreshToken', refreshToken);
}
```

2.2.4 데이터 계층 (Data Layer) - API Client

역할: Backend API 통신 및 데이터 처리

구성:

- `/src/api/api.ts` - 공통 API 요청 함수 (apiFetch, publicApiFetch)
- `/src/api/user.ts` - 사용자 인증 API
- `/src/api/admin.ts` - 관리자 인증 API
- `/src/api/faq.ts` - FAQ API
- `/src/api/qna.ts` - QnA API
- 등 (API 모듈별로 분리)

특징:

- 토큰 자동 갱신 (401 에러 시)
- ErrorCode 기반 에러 처리
- 타임아웃 및 재시도 로직

예시:

```
// src/api/faq.ts
export async function getFaqs(params) {
  return await apiFetch('/api/admin/faq', {
    method: 'GET',
    // 자동 토큰 포함, 401 시 자동 갱신
  });
}
```

2.2.5 라우팅 계층 (Routing Layer) - React Router

역할: URL과 Page 컴포넌트 매핑, 권한 기반 접근 제어

구성:

- `/src/routes/index.ts` - 라우트 정의 (ROUTES 객체)
- `/src/App.tsx` - 라우트 설정 및 Guard 적용
- `/src/components/ProtectedRoute.tsx` - 권한 체크 Guard

특징:

- Public / User / Admin / Common 라우트 분리
- `PrivateRoute` - 일반 사용자 인증 체크
- `AdminProtectedRoute` - 관리자 인증 체크
- 인증 실패 시 로그인 페이지 리다이렉트

예시:

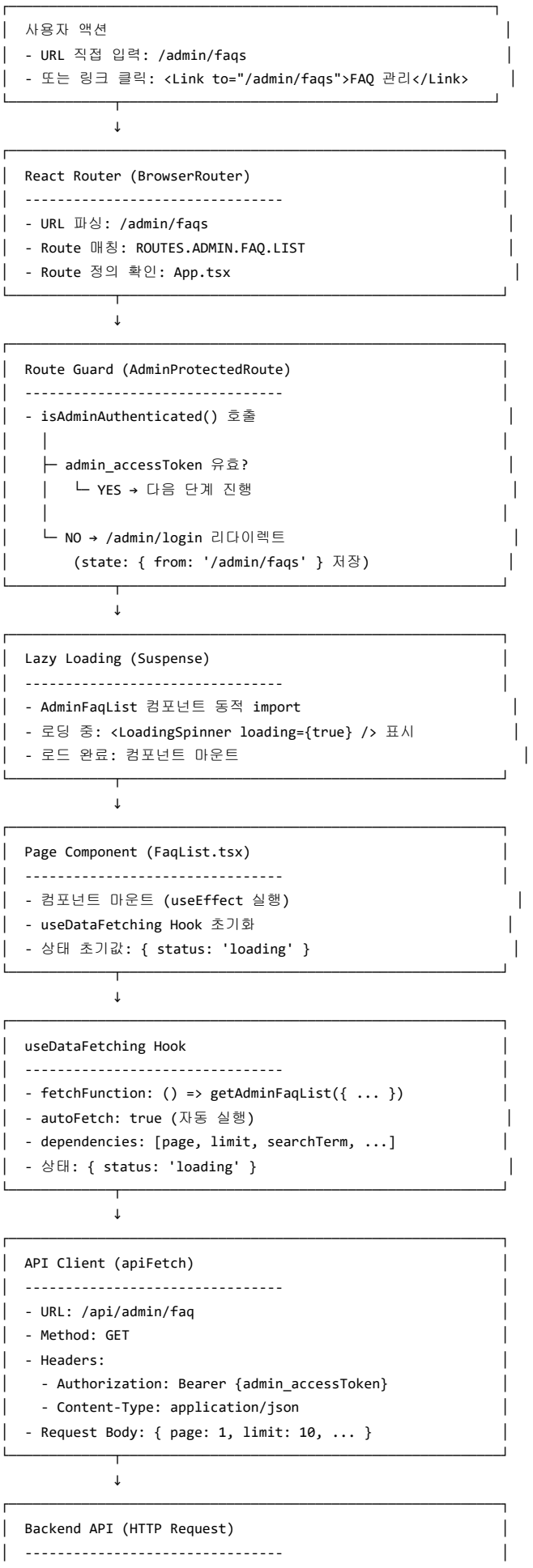
```
// src/App.tsx
<Route
  path="/admin/faqs"
  element={
    <AdminProtectedRoute>
      <FaqList />
    </AdminProtectedRoute>
  }
/>
```

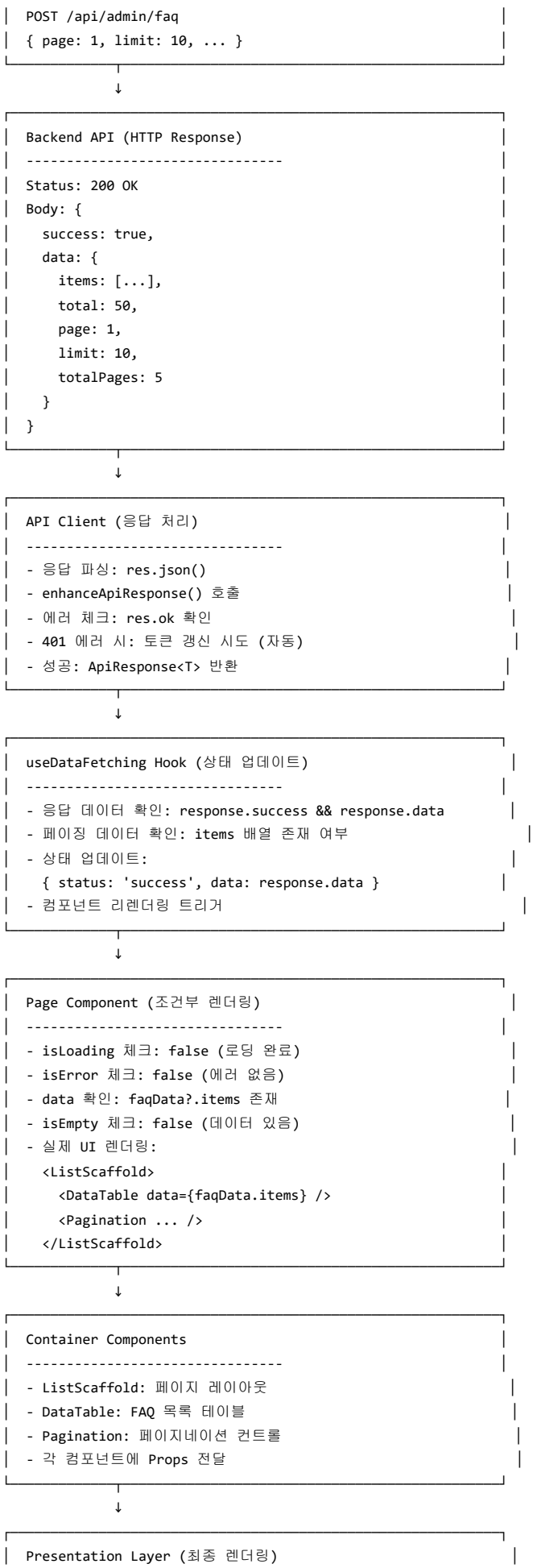
2.3 화면 로드 처리 흐름

모든 화면은 **통일된 처리 흐름**을 따릅니다. 사용자가 URL을 입력하거나 링크를 클릭했을 때부터 화면이 완전히 렌더링될 때까지의 전체 과정을 설명합니다.

2.3.1 전체 처리 흐름 (예: FAQ 관리 페이지)

시나리오: 사용자가 `/admin/faqs` URL에 접근하여 FAQ 목록을 조회하는 경우


사용자 액션
- URL 직접 입력: /admin/faqs
- 또는 링크 클릭: <Link to="/admin/faqs">FAQ 관리</Link>
React Router (BrowserRouter)
- URL 파싱: /admin/faqs
- Route 매칭: ROUTES.ADMIN.FAQ.LIST
- Route 정의 확인: App.tsx
Route Guard (AdminProtectedRoute)
- isAdminAuthenticated() 호출
├─ admin_accessToken 유효?
└─ YES → 다음 단계 진행
└─ NO → /admin/login 리다이렉트
(state: { from: '/admin/faqs' } 저장)
Lazy Loading (Suspense)
- AdminFaqList 컴포넌트 동적 import
- 로딩 중: <LoadingSpinner loading={true} /> 표시
- 로드 완료: 컴포넌트 마운트
Page Component (FaqList.tsx)
- 컴포넌트 마운트 (useEffect 실행)
- useDataFetching Hook 초기화
- 상태 초기값: { status: 'loading' }
useDataFetching Hook
- fetchFunction: () => getAdminFaqList({ ... })
- autoFetch: true (자동 실행)
- dependencies: [page, limit, searchTerm, ...]
- 상태: { status: 'loading' }
API Client (apiFetch)
- URL: /api/admin/faq
- Method: GET
- Headers:
- Authorization: Bearer {admin_accessToken}
- Content-Type: application/json
- Request Body: { page: 1, limit: 10, ... }
Backend API (HTTP Request)


POST /api/admin/faq
{ page: 1, limit: 10, ... }
Backend API (HTTP Response)
Status: 200 OK
Body: {
success: true,
data: {
items: [...],
total: 50,
page: 1,
limit: 10,
totalPages: 5
}
}
API Client (응답 처리)
- 응답 파싱: res.json()
- enhanceApiResponse() 호출
- 에러 체크: res.ok 확인
- 401 에러 시: 토큰 갱신 시도 (자동)
- 성공: ApiResponse<T> 반환
useDataFetching Hook (상태 업데이트)
- 응답 데이터 확인: response.success && response.data
- 페이징 데이터 확인: items 배열 존재 여부
- 상태 업데이트:
{ status: 'success', data: response.data }
- 컴포넌트 리렌더링 트리거
Page Component (조건부 렌더링)
- isLoading 체크: false (로딩 완료)
- isError 체크: false (에러 없음)
- data 확인: faqData?.items 존재
- isEmpty 체크: false (데이터 있음)
- 실제 UI 렌더링:
<ListScaffold>
<DataTable data={faqData.items} />
<Pagination ... />
</ListScaffold>
Container Components
- ListScaffold: 페이지 레이아웃
- DataTable: FAQ 목록 테이블
- Pagination: 페이지네이션 컨트롤
- 각 컴포넌트에 Props 전달
Presentation Layer (최종 렌더링)

```
│  --------------------------------                          │
│  - Material-UI 컴포넌트 렌더링                               │
│  - 테마 적용 (Admin 테마)                                    │
│  - 스타일 적용 (CSS-in-JS)                                   │
│  - 화면 완전히 표시 완료                                     │
└──────────────┬──────────────────────────────────────────────┘
               ↓

┌──────────────────────────────────────────────────────────────┐
│  Browser (화면 표시)                                         │
│  --------------------------------                          │
│  - 사용자가 FAQ 목록 화면 확인                               │
│  - 인터랙션 가능 (클릭, 검색, 페이지 변경 등)                │
└──────────────────────────────────────────────────────────────┘
```

2.3.2 예외 처리 Flow

인증 실패 시:

```
Route Guard (AdminProtectedRoute)
    │
    ├─ isAdminAuthenticated() = false
    │   │
    │   └─ Navigate to /admin/login
    │       (state: { from: '/admin/faqs' } 저장)
    │
    └─ 로그인 성공 후 원래 페이지로 복원
```

API 에러 시:

```
Backend API (HTTP Response)
    │
    ├─ Status: 401 Unauthorized
    │   └─ API Client: 토큰 갱신 시도
    │       ├─ 성공 → 원래 요청 재시도
    │       └─ 실패 → 로그인 페이지 리다이렉트
    │
    ├─ Status: 403 Forbidden
    │   └─ useDataFetching: { status: 'error', error: '접근 권한이 없습니다.' }
    │       └─ Page Component: <ErrorAlert error={error} />
    │
    └─ Status: 500 Internal Server Error
        └─ useDataFetching: { status: 'error', error: '서버 오류가 발생했습니다.' }
            └─ Page Component: <ErrorAlert error={error} />
```

로딩 상태 관리:

```
useDataFetching Hook
    │
    ├─ status: 'loading'
    │   └─ Page Component: <LoadingSpinner loading={true} />
    │
    ├─ status: 'success' && data.items.length > 0
    │   └─ Page Component: 실제 데이터 UI 렌더링
    │
    ├─ status: 'success' && data.items.length === 0
    │   └─ Page Component: <EmptyState message="데이터가 없습니다." />
    │
    └─ status: 'error'
        └─ Page Component: <ErrorAlert error={error} />
```

2.3.3 주요 특징

1. Lazy Loading (코드 스플리팅)

- 관리자 페이지는 `lazy()` 로 동적 import
- 초기 로딩 시간 단축
- `Suspense` 로 로딩 상태 표시

2. 자동 데이터 페칭

- `useDataFetching` 의 `autoFetch: true` 로 마운트 시 자동 실행
- `dependencies` 변경 시 자동 재조회

3. 상태 기반 렌더링

- `isLoading` , `isError` , `isEmpty` 상태에 따라 UI 분기
- 사용자 경험 최적화

4. 토큰 자동 갱신

- API Client에서 401 에러 시 자동 토큰 갱신
- 사용자 개입 없이 세션 유지

2.4 UI 테마 및 스타일 (Material-UI)

Material-UI v5를 사용하여 User와 Admin 사이트의 **독립적인 테마**를 구축합니다.

2.4.1 테마 구분 및 색상 체계

파일: `src/theme/index.ts` , `src/theme/mui.ts`

Frontend는 **User와 Admin 테마를 명확히 분리**합니다:

User 테마 - 밝고 친근한 파란색 계열:

```
user: {
  primary: '#0B5FFF',        // 밝은 파란색
  secondary: '#00B8D9',      // 청록색
  background: '#F5F7FB',     // 연한 회색 배경
  paper: '#FFFFFF',          // 흰색 카드
  text: '#0F172A',           // 짙은 회색 텍스트
  textSecondary: '#475569',  // 중간 회색
}
```

Admin 테마 - 전문적이고 차분한 네이비 계열:

```
admin: {
  primary: '#1E3A8A',        // 짙은 네이비
  secondary: '#3B82F6',      // 밝은 파랑
  background: '#F1F5F9',     // 연한 회색 배경
  paper: '#FFFFFF',          // 흰색 카드
  text: '#111827',           // 짙은 회색 텍스트
  textSecondary: '#6B7280',  // 중간 회색

  // Admin 전용 설정
  spacing: 6,                // 더 compact (User는 8)
  borderRadius: 8,           // 더 작은 radius (User는 10)
  button size: 'small',      // 더 작은 버튼 (User는 'medium')
}
```

Public 페이지 - User 테마 사용:

- 공개 페이지(/, /faq, /qna, /notice)는 **User 테마** 적용
- 일관된 사용자 경험 제공

테마 구분 방법:

```
// src/theme/index.ts
export const getThemeColors = (theme: 'user' | 'admin'): ThemeColors => {
  return THEME_COLORS[theme];
};

// 사용 예시
const colors = getThemeColors('user');     // User 페이지
const colors = getThemeColors('admin');    // Admin 페이지
```

Material-UI 테마 생성:

```
// src/theme/mui.ts
export function createAppTheme(type: 'user' | 'admin', density = 'default') {
  return createTheme({
    ...common,                              // 공통 설정
    ...(type === 'admin' ? adminTheme : userTheme),  // 테마별 설정
    ...densityPreset                        // 밀도 설정 (선택)
  });
}
```

2.4.2 테마별 시각적 차이

요소	User 테마	Admin 테마	목적
Primary 색상	#0B5FFF (밝은 파란색)	#1E3A8A (짙은 네이비)	브랜드 구분
Spacing	8px	6px	Admin은 더 compact
Border Radius	10px	8px	Admin은 더 각진 느낌
Button Size	medium	small	Admin은 정보 밀도 높임
TextField Size	medium	small	Admin은 정보 밀도 높임
느낌	밝고 친근한	전문적이고 차분한	UX 차별화

페이지별 테마 적용:

```
// User 페이지
const colors = getThemeColors('user');

// Admin 페이지
const colors = getThemeColors('admin');

// Public 페이지 (User 테마 사용)
const colors = getThemeColors('user');
```

2.4.3 테마 스타일 유틸리티

파일: `src/theme/index.ts`

```
export const themeStyles = {
  // 페이지 타이틀 스타일 (테마별)
  pageTitle: (theme: 'user' | 'admin') => ({
    color: THEME_COLORS[theme].primary,
    fontWeight: 600,
    borderBottom: `2px solid ${THEME_COLORS[theme].primary}20`
  }),

  // 카드 스타일 (테마별)
  card: (theme: 'user' | 'admin') => ({
    backgroundColor: THEME_COLORS[theme].paper,
    boxShadow: `0 4px 12px ${THEME_COLORS[theme].primary}15`,
    border: `1px solid ${THEME_COLORS[theme].border}`
  }),

  // 버튼 스타일 (테마별)
  primaryButton: (theme: 'user' | 'admin') => ({
    bgcolor: THEME_COLORS[theme].primary,
    color: '#f8f9fa',
    fontWeight: 'bold',
    '&:hover': {
      bgcolor: THEME_COLORS[theme].primary,
      opacity: 0.9,
    }
  }),
};
```

활용 예시:

```
// User 페이지
<ThemedButton theme="user" />        // 파란색 버튼
<ThemedCard theme="user" />          // User 스타일 카드

// Admin 페이지
<ThemedButton theme="admin" />       // 네이비 버튼
<ThemedCard theme="admin" />         // Admin 스타일 카드
```

2.4.4 테마 프리뷰

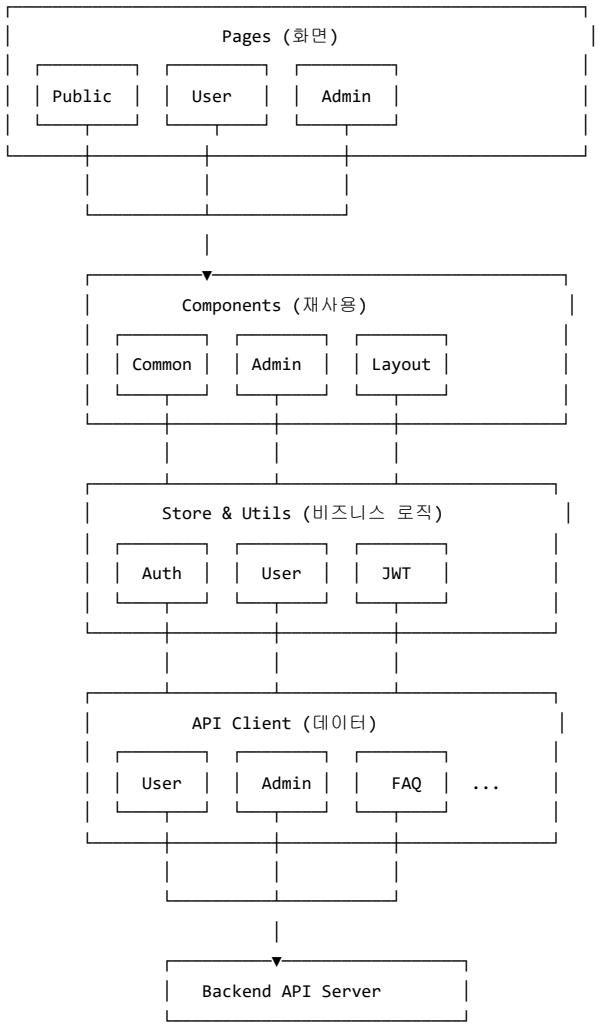
`/theme-preview` 페이지에서 주요 컴포넌트 스타일을 확인할 수 있습니다.

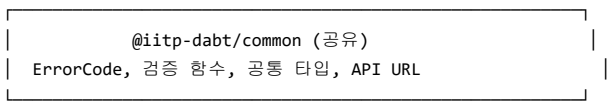
파일: `src/pages/ThemePreview.tsx`

확인 가능 요소:

- 버튼 (primary, secondary, outlined, text)
- 카드 (paper, border)
- 칩 (status, type)
- 입력 필드 (textfield, select)
- 색상 팔레트 (primary, secondary, success, error, warning, info)

2.5 의존성 관계도





의존성 흐름:

1. **Pages** → Components → Store/Utils → API Client → Backend
2. **모든 계층** → Common 패키지 (공유 코드)
3. **React Router** → Pages (라우팅)

3. 디렉토리 구조

3.1 전체 디렉토리 트리

3.1.1 `/src/pages` - 페이지 컴포넌트

Public 페이지 (로그인 불필요):

```
src/pages/
  ThemePreview.tsx                  # UI 테마 프리뷰 페이지

  public/
    OpenApiAbout.tsx                # OpenAPI 소개 페이지
    Privacy.tsx                     # 개인정보 처리방침
    Terms.tsx                       # 이용약관
```

User 페이지 (일반 사용자 인증 필요):

```
src/pages/user/
  Dashboard.tsx                     # 사용자 대시보드
  Home.tsx                          # 홈 페이지 (공개 + 선택적 인증)
  Login.tsx                         # 사용자 로그인
  Register.tsx                      # 사용자 회원가입
  UserProfile.tsx                   # 사용자 프로필 관리

  FaqList.tsx                       # FAQ 목록 (사용자용)

  NoticeList.tsx                    # 공지사항 목록 (사용자용)
  NoticeDetail.tsx                  # 공지사항 상세 (사용자용)

  QnaList.tsx                       # QnA 목록 (사용자용, 공개)
  QnaDetail.tsx                     # QnA 상세 (사용자용, 공개)
  QnaCreate.tsx                     # QnA 생성 (인증 필요)
  QnaHistory.tsx                    # 나의 QnA 히스토리

  OpenApiManagement.tsx             # OpenAPI 키 관리 (사용자용)
```

Admin 페이지 (관리자 인증 필요):

```
src/pages/admin/
  AdminLogin.tsx                  # 관리자 로그인
  AdminDashboard.tsx              # 관리자 대시보드
  AdminProfile.tsx                # 관리자 프로필

  FaqList.tsx                     # FAQ 관리 (목록)
  FaqCreate.tsx                   # FAQ 생성
  FaqDetail.tsx                   # FAQ 상세
  FaqEdit.tsx                     # FAQ 수정

  QnaManage.tsx                   # QnA 관리 (목록)
  QnaDetail.tsx                   # QnA 상세 (관리자용)
  QnaReply.tsx                    # QnA 답변 작성
  QnaEdit.tsx                     # QnA 수정

  NoticeManage.tsx                # 공지사항 관리 (목록)
  NoticeCreate.tsx                # 공지사항 생성
  NoticeDetail.tsx                # 공지사항 상세 (관리자용)
  NoticeEdit.tsx                  # 공지사항 수정

  UserManagement.tsx              # 사용자 관리 (목록)
  UserCreate.tsx                  # 사용자 생성
  UserDetail.tsx                  # 사용자 상세
  UserEdit.tsx                    # 사용자 수정

  OperatorManagement.tsx          # 운영자 관리 (목록, S-ADMIN 전용)
  OperatorCreate.tsx              # 운영자 생성 (S-ADMIN 전용)
  OperatorDetail.tsx              # 운영자 상세 (S-ADMIN 전용)
  OperatorEdit.tsx                # 운영자 수정 (S-ADMIN 전용)

  OpenApiManage.tsx               # OpenAPI 클라이언트 관리
  OpenApiDetail.tsx               # OpenAPI 클라이언트 상세
  OpenApiEdit.tsx                 # OpenAPI 클라이언트 수정
  OpenApiRequests.tsx             # OpenAPI 키 승인 요청 목록
  OpenApiRequestDetail.tsx        # OpenAPI 키 승인 요청 상세

  CodeManagement.tsx              # 코드 관리 (목록, S-ADMIN 전용)
  CodeGroupDetail.tsx             # 코드 그룹 상세 (S-ADMIN 전용)
  CodeCreate.tsx                  # 코드 생성 (S-ADMIN 전용)
  CodeDetail.tsx                  # 코드 상세 (S-ADMIN 전용)
```

3.1.2 `/src/components` - 공통 컴포넌트

레이아웃 컴포넌트:

```
src/components/
  Layout.tsx                      # 전체 레이아웃 (Header + Content + Footer)
  AppBar.tsx                      # 사용자 상단 앱바
  AppBarCommon.tsx                # 공통 앱바 (로직 공유)
  AdminMenuBar.tsx                # 관리자 메뉴바
  Footer.tsx                      # 푸터

  LoginForm.tsx                   # 로그인 폼 (공통)
  ProfileForm.tsx                 # 프로필 폼 (공통)

  ProtectedRoute.tsx              # 권한 체크 Guard (PrivateRoute, AdminProtectedRoute)
  LoadingSpinner.tsx              # 로딩 스피너
  ErrorAlert.tsx                  # 에러 알림

  CommonDialog.tsx                # 공통 다이얼로그
  CommonToast.tsx                 # 공통 토스트
  ToastProvider.tsx               # 토스트 프로바이더
```

Admin 전용 컴포넌트:

```
src/components/admin/
  AdminPageHeader.tsx              # 관리자 페이지 헤더
  SideNav.tsx                      # 관리자 사이드 네비게이션
```

Common UI 컴포넌트:

```
src/components/common/
  DataTable.tsx                    # 데이터 테이블
  TableListBody.tsx                # 테이블 리스트 본문
  CardListBody.tsx                 # 카드 리스트 본문
  ListItemCard.tsx                 # 리스트 아이템 카드
  EmptyState.tsx                   # 빈 상태 표시

  PageHeader.tsx                   # 페이지 헤더
  PageTitle.tsx                    # 페이지 제목
  ListHeader.tsx                   # 리스트 헤더
  ListScaffold.tsx                 # 리스트 스캐폴드
  ListTotal.tsx                    # 리스트 총 개수

  SelectField.tsx                  # 셀렉트 필드
  ThemedButton.tsx                 # 테마 버튼
  ByteLimitHelper.tsx              # 바이트 제한 헬퍼

  StatusChip.tsx                   # 상태 칩
  QnaTypeChip.tsx                  # QnA 타입 칩
  Pagination.tsx                   # 페이지네이션
  ThemedCard.tsx                   # 테마 카드

  ExtendKeyDialog.tsx              # API 키 연장 다이얼로그
```

3.1.3 `/src/api` - API 클라이언트

```
src/api/
  api.ts                           # 공통 API 요청 함수 (apiFetch, publicApiFetch)
  index.ts                         # API 모듈 통합 Export

  user.ts                          # 사용자 인증 및 관리 API
  admin.ts                         # 관리자 인증 API
  account.ts                       # 관리자 계정 관리 API

  common.ts                        # 공통 API (JWT 설정 등)
  commonCode.ts                    # 공통 코드 관리 API

  faq.ts                           # FAQ 관리 API
  qna.ts                           # QnA 관리 API
  notice.ts                        # 공지사항 관리 API
  openApi.ts                       # OpenAPI 키 관리 API
```

3.1.4 `/src/store` - 상태 관리

```
src/store/
  auth.ts                          # 인증 상태 관리 (토큰 저장/조회/갱신)
  user.ts                          # 사용자 정보 관리 (User/Admin 정보 분리)
```

3.1.5 /src/utils - 유틸리티 함수

```
src/utils/
  auth.ts                          # 권한 체크 함수 (isSAdmin, hasContentEditPermission 등)
  jwt.ts                           # JWT 토큰 처리 (검증, 만료 체크, 갱신 판단)
  date.ts                          # 날짜 포맷팅 유틸리티
  openApiStatus.ts                 # OpenAPI 상태 관리 유틸리티
  apiResponseHandler.ts            # API 응답 핸들러
```

3.1.6 /src/routes - 라우팅 설정

```
src/routes/
  index.ts                         # 라우트 정의 (ROUTES, ROUTE_META, RouteUtils)
  guards/
    PublicRoute.tsx                # 공개 라우트 가드 (향후 확장)
```

3.1.7 /src/theme - UI 테마

```
src/theme/
  index.ts                         # 테마 색상 정의 (User/Admin 분리), themeStyles 유틸리티
  mui.ts                           # Material-UI 테마 생성 (createAppTheme)
```

3.1.8 /src/constants - 상수 정의

```
src/constants/
  spacing.ts                       # 페이지 간격 상수 (SPACING, PAGE_SPACING)
  pagination.ts                    # 페이지네이션 기본값 (DEFAULT_PAGE, DEFAULT_LIMIT)
  noticeTypes.ts                   # 공지사항 타입 상수
```

3.1.9 /src/hooks - Custom React Hooks

```
src/hooks/
  useDataFetching.ts               # 데이터 페칭 공통 훅 (로딩, 에러, 빈 상태 관리)
  usePasswordValidation.ts         # 비밀번호 검증 훅
  useQuerySync.ts                  # URL 쿼리 동기화 훅 (검색, 페이징)
  usePagination.ts                 # 페이지네이션 상태 관리 훅
  useInputWithTrim.ts              # 입력 값 trim 처리 훅
  useErrorHandler.ts               # 에러 핸들링 훅
  useCommonCode.ts                 # 공통 코드 조회 훅
```

3.1.10 /src/types - 타입 정의

```
src/types/
  api.ts                           # API 요청/응답 타입 정의
  errorCodes.ts                    # ErrorCode 타입 (Common 패키지 재정의)
```

3.1.11 기타 파일

```
src/
  App.tsx                          # 최상위 App 컴포넌트 (라우트 설정)
  main.tsx                         # 엔트리 포인트 (ReactDOM.render)
  config.ts                        # 환경 변수 설정 (API_BASE_URL 등)
  vite-env.d.ts                    # Vite 환경 변수 타입 정의
  App.css                          # 전역 CSS
  index.css                        # 루트 CSS
  assets/
    react.svg                      # React 로고
```

3.1.12 `/public` - 정적 자산

```
public/
  iitp_cms_logo_img_1.png             # IITP 로고 (메인)
  iitp_cms_logo_img_2.png             # IITP 로고 (서브)
  vite.svg                            # Vite 로고
  index.html                          # HTML 템플릿
```

역할: 빌드 시 `dist/` 로 복사되는 정적 파일

3.2 빌드 및 설정 파일

3.2.1 `vite.config.ts` - Vite 빌드 설정

```typescript
import { defineConfig, loadEnv } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig(({ mode }) => {
  const env = loadEnv(mode, process.cwd(), '')
  return {
    plugins: [react()],
    base: env.VITE_BASE || '/',
    server: {
      port: Number(env.VITE_PORT) || 5173,
    },
    build: {
      outDir: 'dist',
    },
  }
})
```

주요 설정:

- **plugins**: `@vitejs/plugin-react` 사용
- **base**: 서브 경로 배포 지원 (`VITE_BASE` 환경 변수)
- **server.port**: 개발 서버 포트 (기본 5173)
- **build.outDir**: 빌드 출력 디렉토리 (`dist/`)

3.2.2 `tsconfig.json` - TypeScript 설정

```json
{
  "files": [],
  "references": [
    { "path": "../packages/common" },
    { "path": "./tsconfig.app.json" },
    { "path": "./tsconfig.node.json" }
  ],
  "compilerOptions": {
    "baseUrl": ".",
    "paths": {
      "packages/common/*": ["../packages/common/*"]
    }
  }
}
```

주요 설정:

- **references**: Project References로 Common 패키지 참조
- **paths**: `@iitp-dabt/common` 경로 매핑

3.2.3 `package.json` - 의존성 및 스크립트

주요 스크립트:

```
{
  "scripts": {
    "dev": "vite",
    "build": "rimraf dist && tsc -b && vite build && node scripts/build-info.js",
    "build:clean": "rimraf dist && tsc -b && vite build && node scripts/build-info.js",
    "preview": "vite preview",
    "prebuild": "node scripts/build-info.js"
  }
}
```

- **dev**: 개발 서버 실행
- **build**: 프로덕션 빌드 (빌드 정보 생성 포함)
- **build:clean**: 클린 빌드
- **preview**: 빌드 결과물 미리보기

3.2.4 `scripts/` - 빌드 스크립트

```
scripts/
  build-info.js                    # 빌드 정보 생성 (version, buildDate)
  build.sh                         # 빌드 스크립트 (Linux/Mac)
  setup.sh                         # 초기 설정 스크립트
```

scripts/build-info.js:

```
// 빌드 정보 생성 스크립트
const buildInfo = {
  version: pkg.version,
  buildDate: getLocalDateTimeString()
};

fs.writeFileSync(path.join(distPath, 'build-info.json'),
  JSON.stringify(buildInfo, null, 2));
```

역할: 빌드 버전 및 빌드 시각을 `dist/build-info.json` 으로 출력

3.2.5 루트 설정 파일

```
fe/
  .env                             # 환경 변수 설정 (로컬, git ignore)
  .env.sample                      # 환경 변수 샘플 파일
  package.json                     # 의존성 및 스크립트
  package-lock.json                # 의존성 잠금 파일
  vite.config.ts                   # Vite 빌드 설정
  tsconfig.json                    # TypeScript 루트 설정 (Project References)
  tsconfig.app.json                # App TypeScript 설정
  tsconfig.node.json               # Node TypeScript 설정 (Vite 빌드 스크립트용)
  eslint.config.js                 # ESLint 설정
  index.html                       # HTML 엔트리 포인트
  README.md                        # 프로젝트 설명
```

3.2.6 `public/`

```
public/
  iitp_cms_logo_img_1.png          # IITP 로고 (메인)
  iitp_cms_logo_img_2.png          # IITP 로고 (서브)
  vite.svg                         # Vite 로고
  index.html                       # HTML 템플릿 (빌드 시 사용)
```

역할: 빌드 시 `dist/` 로 복사되는 정적 파일 (로고, 아이콘 등)

4. 인증 및 권한 체계

Frontend는 **JWT 기반 인증**과 **Role-Based 권한 체크**를 합니다.

> **중요**: Frontend의 인증 및 권한 체크는 **UX**이며, 실제 보안은 Backend에서 담당합니다.

4.1 JWT 토큰 관리

4.1.1 토큰 검증 로직

파일: `src/utils/jwt.ts`

주요 함수:

```
import { jwtDecode } from 'jwt-decode';

// 토큰 형식 검증
export function isValidTokenFormat(token: string): boolean {
  if (!token || typeof token !== 'string') return false;
  const parts = token.split('.');
  return parts.length === 3; // header.payload.signature
}

// 토큰 정보 추출
export function extractTokenInfo(token: string): TokenInfo | null {
  try {
    const decoded = jwtDecode(token) as any;
    if (!decoded || !decoded.exp || !decoded.iat) return null;

    return {
      exp: decoded.exp,
      iat: decoded.iat,
      expiresIn: JWT_CONFIG.accessTokenExpiresIn,
    };
  } catch {
    return null;
  }
}
```

4.1.2 토큰 만료 확인

```
// 토큰 만료 여부 확인
export function isTokenExpired(token: string): boolean {
  try {
    const decoded = jwtDecode(token) as any;
    if (!decoded || !decoded.exp) return true;

    const currentTime = Math.floor(Date.now() / 1000);
    return decoded.exp < currentTime;
  } catch {
    return true;
  }
}

// 토큰 만료까지 남은 시간 (초)
export function getTokenTimeRemaining(token: string): number {
  try {
    const decoded = jwtDecode(token) as any;
    if (!decoded || !decoded.exp) return -1;

    const currentTime = Math.floor(Date.now() / 1000);
    return decoded.exp - currentTime;
  } catch {
    return -1;
  }
}
```

4.1.3 토큰 갱신 판단 (만료 5분 전)

```
// 토큰 갱신이 필요한지 확인 (기본값: 만료 5분 전)
export function shouldRefreshToken(
  token: string,
  bufferSeconds: number = 300  // 5분
): boolean {
  const timeRemaining = getTokenTimeRemaining(token);
  return timeRemaining > 0 && timeRemaining <= bufferSeconds;
}
```

갱신 판단 로직:

- 만료까지 **5분 이하 남았을 때** 갱신 필요
- 이미 만료된 경우도 갱신 시도
- `bufferSeconds` 파라미터로 갱신 시점 조정 가능

4.2 인증 상태 관리

4.2.1 토큰 저장/조회

파일: `src/store/auth.ts`

```
// User/Admin 독립적 저장을 위한 prefix 상수
const USER_PREFIX = 'user_';
const ADMIN_PREFIX = 'admin_';

// 현재 활성 사용자 타입에 맞는 prefix 자동 반환
function getCurrentPrefix(): string {
  const userType = getUserType();
  return userType === 'A' ? ADMIN_PREFIX : USER_PREFIX;
}

// 토큰 저장
export function saveTokens(accessToken: string, refreshToken: string) {
  if (!isValidTokenFormat(accessToken) || !isValidTokenFormat(refreshToken)) {
    console.warn('Invalid token format detected');
    return;
  }

  const prefix = getCurrentPrefix();
  localStorage.setItem(prefix + 'accessToken', accessToken);
  localStorage.setItem(prefix + 'refreshToken', refreshToken);
}

// Access Token 가져오기
export function getAccessToken(): string | null {
  const prefix = getCurrentPrefix();
  const token = localStorage.setItem(prefix + 'accessToken');
  if (!token || !isValidTokenFormat(token)) return null;
  return token;
}
```

저장 위치: `LocalStorage`

- User 토큰: `user_accessToken` , `user_refreshToken`
- Admin 토큰: `admin_accessToken` , `admin_refreshToken`

4.2.2 토큰 만료 확인

```
// Access Token 만료 여부
export function isAccessTokenExpired(): boolean {
  const token = getAccessToken();
  return !token || isTokenExpired(token);
}

// Refresh Token 만료 여부
export function isRefreshTokenExpired(): boolean {
  const token = getRefreshToken();
  return !token || isTokenExpired(token);
}

// Access Token 갱신 필요 여부
export function shouldRefreshAccessToken(): boolean {
  const token = getAccessToken();
  return token ? shouldRefreshToken(token) : true;
}
```

4.2.3 토큰 갱신 조건 및 Flow

갱신 조건:

1. **Access Token이 만료 5분 전**일 때
2. **Access Token이 만료**되었고 Refresh Token이 유효할 때

갱신 Flow:

```
// 토큰 상태 확인 및 갱신 (API 요청 전 호출)
export async function ensureValidToken(): Promise<string | null> {
  validateAndCleanTokens();  // 유효성 검사 및 정리

  const accessToken = getAccessToken();
  const refreshToken = getRefreshToken();

  // 토큰이 없으면 null 반환
  if (!accessToken) {
    // Access 없음 → Refresh로 갱신 시도
    if (refreshToken && !isTokenExpired(refreshToken)) {
      return await tryRefreshToken(refreshToken);
    }
    return null;
  }

  // Access Token이 유효하고 갱신 필요 없으면 그대로 사용
  if (!isTokenExpired(accessToken) && !shouldRefreshToken(accessToken)) {
    return accessToken;
  }

  // Access 만료 또는 만료 임박 → Refresh로 갱신 시도
  if (refreshToken && !isTokenExpired(refreshToken)) {
    return await tryRefreshToken(refreshToken);
  }

  return null;  // 갱신 실패
}

// Refresh Token으로 Access/Refresh 재발급 시도
async function tryRefreshToken(refreshToken: string): Promise<string | null> {
  try {
    const userType = getUserType();
    const url = userType === 'A'
      ? FULL_API_URLS.AUTH.ADMIN.REFRESH
      : FULL_API_URLS.AUTH.USER.REFRESH;

    const res = await fetch(`${API_BASE_URL}${url}`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ refreshToken })
    });

    if (!res.ok) throw new Error(`Refresh failed: ${res.status}`);

    const data = await res.json();
    const newAccess = data?.data?.token || data?.token;
    const newRefresh = data?.data?.refreshToken || data?.refreshToken;

    if (newAccess && newRefresh &&
        isValidTokenFormat(newAccess) &&
        isValidTokenFormat(newRefresh)) {
      saveTokens(newAccess, newRefresh);
      return newAccess;
    }

    throw new Error('Invalid refresh response');
  } catch (error) {
    console.error('Token refresh failed:', error);
    removeTokens();
    clearLoginInfo();
    return null;
  }
}
```

Flow 다이어그램:

```
API 요청 전
    │
    ▼
ensureValidToken() 호출
    │
    ├─ Access Token 없음?
    │   └─ Refresh Token 있고 유효? → 갱신 시도
    │       ├─ 성공 → 새 Access Token 반환
    │       └─ 실패 → null 반환 (로그인 필요)
    │
    ├─ Access Token 있고 유효? (만료 5분 초과)
    │   └─ 그대로 사용
    │
    └─ Access Token 만료 또는 만료 임박? (5분 이내)
        └─ Refresh Token으로 갱신 시도
            ├─ 성공 → 새 Access Token 반환
            └─ 실패 → null 반환 (로그인 필요)
```

4.2.4 인증 상태 확인

```typescript
// 기본 인증 상태 확인
export function isAuthenticated(): boolean {
  const accessToken = getAccessToken();
  const refreshToken = getRefreshToken();

  return (!!accessToken && !isTokenExpired(accessToken)) ||
         (!!refreshToken && !isTokenExpired(refreshToken));
}

// 일반 사용자 인증 상태 확인
export function isUserAuthenticated(): boolean {
  const userAccessToken = localStorage.getItem('user_accessToken');
  const userRefreshToken = localStorage.getItem('user_refreshToken');

  const hasValidUserToken =
    (userAccessToken && !isTokenExpired(userAccessToken)) ||
    (userRefreshToken && !isTokenExpired(userRefreshToken));

  const userInfo = localStorage.getItem('user_userInfo');
  return !!(hasValidUserToken && userInfo);
}

// 관리자 인증 상태 확인
export function isAdminAuthenticated(): boolean {
  const adminAccessToken = localStorage.getItem('admin_accessToken');
  const adminRefreshToken = localStorage.getItem('admin_refreshToken');

  const hasValidAdminToken =
    (adminAccessToken && !isTokenExpired(adminAccessToken)) ||
    (adminRefreshToken && !isTokenExpired(adminRefreshToken));

  const adminInfo = localStorage.getItem('admin_userInfo');
  return !!(hasValidAdminToken && adminInfo);
}
```

4.3 사용자 정보 관리

4.3.1 사용자 정보 저장/조회

파일: `src/store/user.ts`

```
interface UserInfo {
  userId: number;
  email: string;
  name: string;
  userType: 'U' | 'A';
  role?: string;       // 관리자의 경우 role 정보
  roleName?: string;   // 관리자의 경우 role 이름
}

// 사용자 정보 저장
export function saveUserInfo(userInfo: UserInfo) {
  const prefix = getCurrentPrefix(userInfo.userType);
  localStorage.setItem(prefix + 'userInfo', JSON.stringify(userInfo));
}

// 사용자 정보 가져오기
export function getUserInfo(): UserInfo | null {
  const prefix = getCurrentPrefix();
  const userInfoStr = localStorage.getItem(prefix + 'userInfo');
  if (!userInfoStr) return null;

  try {
    return JSON.parse(userInfoStr);
  } catch (error) {
    console.error('Failed to parse user info:', error);
    return null;
  }
}
```

4.3.2 Admin Role 조회

```
// 권한 체크용 Admin Role 반환 (userInfo.role 사용)
export function getAdminRole(): string {
  const adminInfoStr = localStorage.getItem('admin_userInfo');
  if (!adminInfoStr) return '';

  try {
    const adminInfo = JSON.parse(adminInfoStr);
    return adminInfo?.role || '';
  } catch (error) {
    console.error('Failed to parse admin info:', error);
    return '';
  }
}

// 화면 표시용 Admin Role Name 반환
export function getAdminRoleName(): string {
  const adminInfoStr = localStorage.getItem('admin_userInfo');
  if (!adminInfoStr) return '관리자';

  try {
    const adminInfo = JSON.parse(adminInfoStr);
    return adminInfo?.roleName || '관리자';
  } catch (error) {
    console.error('Failed to parse admin info:', error);
    return '관리자';
  }
}
```

4.3.3 로그인 정보 저장/삭제

```
// 로그인 시 사용자 정보와 토큰을 함께 저장
export function saveLoginInfo(
  userInfo: UserInfo,
  accessToken: string,
  refreshToken: string
) {
  saveUserInfo(userInfo);
  saveTokens(accessToken, refreshToken);
}

// 현재 활성 사용자 타입의 로그인 정보만 제거
export function clearLoginInfo() {
  removeUserInfo();  // 현재 타입의 사용자 정보만 제거
  removeTokens();    // 현재 타입의 토큰만 제거
}

// 특정 사용자 타입의 로그인 정보만 제거
export function clearLoginInfoByType(userType: 'U' | 'A') {
  removeUserInfoByType(userType);
  removeTokensByType(userType);
}

// 모든 타입의 로그인 정보 완전 제거
export function clearAllLoginInfo() {
  removeAllUserInfo();
  removeAllTokens();
}
```

4.4 User/Admin 독립 세션 관리 (핵심 로직)

Frontend는 **User와 Admin의 독립적인 동시 로그인**을 지원합니다.

4.4.1 LocalStorage prefix 분리 (`user_` , `admin_`)

저장 구조:

```
LocalStorage:
  user_accessToken       # 일반 사용자 Access Token
  user_refreshToken      # 일반 사용자 Refresh Token
  user_userInfo          # 일반 사용자 정보

  admin_accessToken      # 관리자 Access Token
  admin_refreshToken     # 관리자 Refresh Token
  admin_userInfo         # 관리자 정보
```

장점:

- **독립적 세션**: User와 Admin 세션이 서로 영향을 주지 않음
- **동시 로그인**: 하나의 브라우저에서 User와 Admin 동시 로그인 가능
- **세션 전환**: 로그아웃 없이 User ↔ Admin 전환 가능

4.4.2 동시 로그인 지원 메커니즘

현재 활성 사용자 타입 자동 판단:

```
// 현재 활성 사용자 타입 자동 판단 (Admin 우선)
export function getUserType(): 'U' | 'A' | null {
  // Admin 정보 먼저 확인 (우선순위)
  const adminInfo = localStorage.getItem('admin_userInfo');
  if (adminInfo) {
    try {
      const parsed = JSON.parse(adminInfo);
      if (parsed && parsed.userType === 'A') return 'A';
    } catch {}
  }

  // User 정보 확인
  const userInfo = localStorage.getItem('user_userInfo');
  if (userInfo) {
    try {
      const parsed = JSON.parse(userInfo);
      if (parsed && parsed.userType === 'U') return 'U';
    } catch {}
  }

  return null;  // 둘 다 없으면 null
}
```

우선순위: Admin > User

- Admin으로 로그인되어 있으면 Admin 세션 사용
- Admin이 없고 User만 있으면 User 세션 사용
- 둘 다 없으면 `null` (로그인 필요)

4.4.3 자동 사용자 타입 판단 (Admin 우선)

prefix 자동 결정:

```
function getCurrentPrefix(): string {
  const userType = getUserType();
  return userType === 'A' ? ADMIN_PREFIX : USER_PREFIX;
}
```

활용:

- `saveTokens()` : 현재 활성 타입에 맞게 토큰 저장
- `getAccessToken()` : 현재 활성 타입의 토큰 조회
- `saveUserInfo()` : 현재 활성 타입의 사용자 정보 저장

4.4.4 세션 전환 Flow

시나리오 1: User → Admin 전환

```
1. User로 로그인 상태
   - user_accessToken, user_userInfo 존재

2. Admin 로그인 (/admin/login)
   - admin_accessToken, admin_userInfo 생성

3. getUserType() 호출
   - Admin 우선 → 'A' 반환

4. 이후 모든 API 요청은 admin_ prefix 사용
   - User 세션은 유지됨 (로그아웃 불필요)
```

시나리오 2: Admin → User 전환

```
1. Admin으로 로그인 상태
   - admin_accessToken, admin_adminInfo 존재

2. Admin 로그아웃 (clearLoginInfoByType('A'))
   - admin_ prefix 데이터 삭제

3. getUserType() 호출
   - Admin 없음 → User 확인 → 'U' 반환

4. User 세션으로 자동 전환
```

시나리오 3: 완전 로그아웃

```
1. User와 Admin 모두 로그인 상태

2. 완전 로그아웃 (clearAllLoginInfo())
   - user_ prefix 데이터 삭제
   - admin_ prefix 데이터 삭제

3. getUserType() 호출
   - null 반환 (로그인 필요)
```

4.4.5 브라우저 저장 데이터 종합

Frontend는 **LocalStorage**와 **SessionStorage**를 사용하여 인증 및 사용자 정보를 저장합니다.

LocalStorage (영구 저장):

키 이름	데이터 형식	용도	예시 값
`user_accessToken`	JWT 문자열	일반 사용자	`eyJhbGciOiJIUzI1NiIs...`

키 이름	데이터 형식	용도	예시 값
		Access Token	
`user_refreshToken`	JWT 문자열	일반 사용자 Refresh Token	`eyJhbGciOiJIUzI1NiIs...`
`user_userInfo`	JSON 문자열	일반 사용자 정보	`{"userId":1,"email":"user@example.com","name":"홍길동","userType":"U"}`
`admin_accessToken`	JWT 문자열	관리자 Access Token	`eyJhbGciOiJIUzI1NiIs...`
`admin_refreshToken`	JWT 문자열	관리자 Refresh Token	`eyJhbGciOiJIUzI1NiIs...`
`admin_userInfo`	JSON 문자열	관리자 정보 (role 포함)	`{"userId":1,"email":"admin@example.com","name":"관리자","userType":"A","role":"S-ADMIN","roleName":"Su`

SessionStorage (탭 닫으면 삭제):

키 이름	데이터 형식	용도
`returnTo`	문자열 (URL 경로)	인증 실패 시 원래 페이지 경로 저장 (로그인 후 자동 복원)

저장 데이터 특징:

- **User와 Admin 완전 분리**: prefix로 구분하여 독립 세션 지원
- **로그아웃 시 선택적 삭제**:
  - `clearLoginInfoByType('U')` → user_ prefix만 삭제
  - `clearLoginInfoByType('A')` → admin_ prefix만 삭제
  - `clearAllLoginInfo()` → 모든 데이터 삭제
- **SessionStorage 활용**: 페이지 복원을 위한 임시 데이터 (보안 강화)
- **자동 정리**: 유효하지 않은 토큰은 자동 제거 (`validateAndCleanTokens`)

보안 고려사항:

- LocalStorage는 XSS 공격에 취약 (JavaScript로 접근 가능)
- 토큰 형식 검증 (`isValidTokenFormat`) 적용
- 토큰 만료 체크 (`isTokenExpired`) 적용
- HTTPS 통신 필수 (프로덕션)

4.5 권한 체크 유틸리티

4.5.1 역할 기반 권한 체크 함수 (8개)

파일: `src/utils/auth.ts`

```typescript
import { CODE_SYS_ADMIN_ROLES } from '@iitp-dabt/common';

// 1. S-Admin 권한 확인 (최고 권한)
export function isSAdmin(adminRole: string | null): boolean {
  return adminRole === CODE_SYS_ADMIN_ROLES.SUPER_ADMIN;
}

// 2. 일반 Admin 권한 확인 (ADMIN, EDITOR, VIEWER 포함)
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;
}

// 3. 콘텐츠 편집 권한 확인 (S-ADMIN, ADMIN, EDITOR)
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;
}

// 4. 읽기 권한 확인 (모든 관리자)
export function hasReadPermission(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;
}

// 5. 운영자 계정 관리 권한 확인 (S-ADMIN만)
export function hasAccountManagementPermission(adminRole: string | null): boolean {
  return adminRole === CODE_SYS_ADMIN_ROLES.SUPER_ADMIN;
}

// 6. 사용자 계정 조회 권한 확인 (모든 관리자)
export function hasUserAccountReadPermission(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;
}

// 7. 사용자 계정 편집 권한 확인 (S-ADMIN, ADMIN만)
export function hasUserAccountEditPermission(adminRole: string | null): boolean {
  return adminRole === CODE_SYS_ADMIN_ROLES.SUPER_ADMIN
      || adminRole === CODE_SYS_ADMIN_ROLES.ADMIN;
}

// 8. 시스템 설정 권한 확인 (S-ADMIN만)
export function hasSystemConfigPermission(adminRole: string | null): boolean {
  return adminRole === CODE_SYS_ADMIN_ROLES.SUPER_ADMIN;
}
```

권한 매트릭스:

기능	VIEWER	EDITOR	ADMIN	S-ADMIN
콘텐츠 조회 (FAQ, QnA, Notice)	O	O	O	O
콘텐츠 편집 (생성/수정/삭제)	X	O	O	O
사용자 계정 조회	O	O	O	O
사용자 계정 편집	X	X	O	O

기능	VIEWER	EDITOR	ADMIN	S-ADMIN
운영자 계정 관리	X	X	X	O
코드 관리	X	X	X	O
시스템 설정	X	X	X	O

4.5.2 메뉴 접근 권한 체크

```
export function hasMenuAccess(adminRole: string | null, menuName: string): boolean {
  switch (menuName) {
    case 'dashboard':
    case 'openapi':
    case 'qna':
    case 'faq':
    case 'notice':
    case 'user-management':
      return hasReadPermission(adminRole);

    case 'operator-management':
    case 'code-management':
      return hasAccountManagementPermission(adminRole);

    default:
      return hasReadPermission(adminRole);
  }
}
```

활용: 사이드 네비게이션 메뉴 표시/숨김 제어

4.5.3 버튼/액션 권한 체크

```
export function hasActionPermission(adminRole: string | null, actionType: string): boolean {
  switch (actionType) {
    case 'create':
    case 'update':
    case 'delete':
      return hasContentEditPermission(adminRole);

    case 'user-create':
    case 'user-update':
    case 'user-delete':
      return hasUserAccountEditPermission(adminRole);

    case 'operator-create':
    case 'operator-update':
    case 'operator-delete':
      return hasAccountManagementPermission(adminRole);

    case 'code-create':
    case 'code-update':
    case 'code-delete':
      return hasAccountManagementPermission(adminRole);

    default:
      return hasReadPermission(adminRole);
  }
}
```

활용: 버튼 활성화/비활성화, 경고 메시지 표시

4.6 라우트 가드

4.6.1 `PrivateRoute` - 일반 사용자 인증 체크

파일: `src/components/ProtectedRoute.tsx`

```tsx
import { Navigate, useLocation } from 'react-router-dom';
import { useEffect } from 'react';
import { validateAndCleanTokens, isUserAuthenticated } from '../store/auth';
import { ROUTES } from '../routes';

// 인증 보호 라우트 컴포넌트 (일반 사용자용)
export function PrivateRoute({ children }: { children: React.ReactNode }) {
  const isLoggedIn = isUserAuthenticated();
  const location = useLocation();

  // 토큰 유효성 검사 및 정리
  useEffect(() => {
    validateAndCleanTokens();
  }, []);

  if (!isLoggedIn) {
    return <Navigate to={ROUTES.PUBLIC.LOGIN} state={{ from: location }} replace />;
  }

  return <>{children}</>;
}
```

동작:

1. `isUserAuthenticated()` 호출로 User 토큰 유효성 확인
2. 유효하지 않으면 `/login` 으로 리다이렉트
3. `location.state` 에 원래 페이지 경로 저장 (로그인 후 복원)

적용 페이지:

- `/dashbd` - 사용자 대시보드
- `/profile` - 프로필 관리
- `/user/qna/create` - QnA 생성
- `/user/openapi` - OpenAPI 키 관리

4.6.2 `AdminProtectedRoute` - 관리자 인증 체크

```tsx
// 관리자 보호 라우트 컴포넌트
export function AdminProtectedRoute({ children }: { children: React.ReactNode }) {
  const isLoggedIn = isAdminAuthenticated();
  const location = useLocation();

  // 토큰 유효성 검사 및 정리
  useEffect(() => {
    validateAndCleanTokens();
  }, []);

  if (!isLoggedIn) {
    return <Navigate to={ROUTES.ADMIN.LOGIN} state={{ from: location }} replace />;
  }

  return <>{children}</>;
}
```

동작:

1. `isAdminAuthenticated()` 호출로 Admin 토큰 유효성 확인

2. 유효하지 않으면 `/admin/login` 으로 리다이렉트
3. `location.state` 에 원래 페이지 경로 저장

적용 페이지:

- `/admin/dashbd` - 관리자 대시보드
- `/admin/faqs/*` - FAQ 관리
- `/admin/qnas/*` - QnA 관리
- `/admin/notices/*` - 공지사항 관리
- `/admin/users/*` - 사용자 관리
- `/admin/operators/*` - 운영자 관리
- `/admin/openapi/*` - OpenAPI 관리
- `/admin/code/*` - 코드 관리

4.6.3 인증 실패 시 리다이렉트

Flow:

```
사용자가 보호된 페이지 접근
    │
    ▼
PrivateRoute / AdminProtectedRoute
    │
    ├─ 인증 성공?
    │   └─ 페이지 렌더링
    │
    └─ 인증 실패?
        └─ 로그인 페이지로 리다이렉트
            ├─ state: { from: location }
            └─ 로그인 성공 후 원래 페이지로 복원
```

로그인 후 복원 예시:

```tsx
// Login.tsx
const location = useLocation();
const from = location.state?.from?.pathname || '/dashbd';

const handleLoginSuccess = () => {
  navigate(from, { replace: true });
};
```

5. API 클라이언트

5.1 API 클라이언트 구조 및 공통 로직

Frontend의 모든 Backend API 통신은 `src/api/` 디렉토리의 모듈을 통해 이루어집니다.

주요 특징:

- **통합 요청 함수**: `apiFetch()` (인증 필요), `publicApiFetch()` (공개)
- **자동 토큰 관리**: 토큰 갱신, 만료 체크
- **ErrorCode 기반 처리**: 사용자 친화적 메시지 변환
- **타임아웃 및 재시도**: 네트워크 오류 처리

5.1.1 `apiFetch()` vs `publicApiFetch()` 차이

항목	`apiFetch()`	`publicApiFetch()`
토큰 필수 여부	필수 (없으면 null 반환)	선택 (없어도 진행)
토큰 갱신	요청 전 자동 갱신	401 에러 시 재시도
사용 페이지	User/Admin 인증 페이지	공개 페이지 (FAQ, QnA, Notice)
Authorization 헤더	항상 포함	토큰 있을 때만 포함

5.1.2 토큰 자동 갱신 (401 에러 시)

`apiFetch()` 갱신 Flow:

```
API 요청 전 → ensureValidToken() 호출
  ├─ Access Token 유효 (만료 5분 초과)
  │   └─ 그대로 사용
  └─ Access Token 만료 또는 만료 임박 (5분 이내)
      └─ Refresh Token으로 갱신 시도
          ├─ 성공 → 새 Access Token 사용
          └─ 실패 → null 반환 (API 요청 중단)
```

`publicApiFetch()` 갱신 Flow:

```
API 요청 → 401 에러 발생 시
  └─ Refresh Token으로 갱신 시도
      ├─ 성공 → 동일 요청 재시도 (1회만)
      └─ 실패 → 에러 응답 반환
```

5.1.3 정상 응답 처리 (`ApiResponse<T>` 구조)

타입 정의: `src/types/api.ts`

```typescript
export interface ApiResponse<T = any> {
  success: boolean;
  data?: T;
  errorCode?: number;
  errorMessage?: string;

  // FE 전용 확장 필드
  showPopup?: boolean;       // 팝업 표시 필요 여부
  redirectTo?: string;       // 리다이렉트 URL
  autoLogout?: boolean;      // 자동 로그아웃 필요 여부
  details?: any;             // 추가 상세 정보
}
```

5.1.4 에러 응답 처리 (ErrorCode → 사용자 친화적 메시지)

ErrorCode 기반 메시지 변환 (`src/api/api.ts`):

ErrorCode	사용자 메시지
`UNAUTHORIZED` (40101)	인증이 필요합니다. 다시 로그인해주세요.
`TOKEN_EXPIRED` (40102)	로그인 세션이 만료되었습니다. 다시 로그인해주세요.
`INVALID_TOKEN` (40103)	유효하지 않은 인증 정보입니다. 다시 로그인해주세요.
`ACCESS_DENIED` (40301)	접근 권한이 없습니다.
`USER_NOT_FOUND` (40401)	사용자를 찾을 수 없습니다.
`LOGIN_FAILED` (40001)	로그인에 실패했습니다. 아이디와 비밀번호를 확인해주세요.
`NETWORK_ERROR` (50301)	네트워크 오류가 발생했습니다.
`REQUEST_TIMEOUT` (50801)	요청 시간이 초과되었습니다.

응답 강화 (`src/utils/apiResponseHandler.ts`):

- `showPopup` : 팝업 표시 필요 여부 자동 판단
- `redirectTo` : 리다이렉트 URL 자동 생성 (User/Admin 타입별)
- `autoLogout` : 자동 로그아웃 필요 여부 판단

5.1.5 재시도 로직 (타임아웃, 네트워크 오류)

타임아웃 설정:

- 기본 타임아웃: `10초` (`API_TIMEOUT`)
- 커스텀 타임아웃: `options.timeoutMs` 로 조정 가능

재시도 정책:

- **401 에러 (인증 실패)**: 토큰 갱신 후 1회 재시도
- **타임아웃**: 재시도 없음 (에러 메시지 표시)
- **네트워크 오류**: 재시도 없음 (에러 메시지 표시)

5.2 API 모듈 목록 (테이블 형태)

파일	주요 함수	설명	사용처
`api.ts`	`apiFetch` , `publicApiFetch`	공통 API 요청 함수	모든 API 모듈
`user.ts`	`login` , `register` , `getProfile` , `updateProfile` , `changePassword`	사용자 인증 및 관리	로그인, 회원가입, 프로필 페이지
`admin.ts`	`loginAdmin` , `refreshAdminToken` , `getAdminProfile`	관리자 인증	관리자 로그인, 프로필 페이지
`account.ts`	`getAdminAccounts` , `createAdminAccount` , `updateAdminAccount` , `deleteAdminAccount`	관리자 계정 관리	운영자 관리 페이지 (S-ADMIN)
`common.ts`	`getJwtConfig` , `getSystemInfo`	공통 API	JWT 설정 조회, 시스템 정보
`commonCode.ts`	`getCommonCodes` , `getCodesByGroup` , `createCode` , `updateCode` , `deleteCode`	공통 코드 관리	코드 관리 페이지 (S-ADMIN)
`faq.ts`	`getFaqs` , `getFaqById` , `createFaq` , `updateFaq` , `deleteFaq` , `batchDeleteFaq`	FAQ 관리	FAQ 목록, 상세, 생성, 수정 페이지
`qna.ts`	`getQnas` , `getQnaById` , `createQna` , `updateQna` , `deleteQna` , `replyQna`	QnA 관리	QnA 목록, 상세, 생성, 답변 페이지

파일	주요 함수	설명	사용처
`notice.ts`	`getNotices`, `getNoticeById`, `createNotice`, `updateNotice`, `deleteNotice`	공지사항 관리	공지사항 목록, 상세, 생성, 수정 페이지
`openApi.ts`	`getOpenApiClients`, `approveOpenApiKey`, `rejectOpenApiKey`, `extendOpenApiKey`	OpenAPI 키 관리	OpenAPI 클라이언트 관리, 키 승인 페이지

특징:

- **일관된 네이밍**: `get*`, `create*`, `update*`, `delete*`, `batchDelete*`
- **타입 안전**: TypeScript 제네릭을 통한 타입 추론
- **에러 핸들링**: 모든 함수는 `ApiResponse<T>` 반환

6. 라우팅 구조

6.1 라우트 정의

6.1.1 ROUTES 객체 구조 (PUBLIC, USER, ADMIN, COMMON)

파일: src/routes/index.ts

Frontend는 **4단계 라우트 분류**를 사용합니다:

1. PUBLIC - 공개 페이지 (로그인 불필요)

```
HOME: '/'
THEME_PREVIEW: '/theme-preview'
NOTICE: '/notice'
NOTICE_DETAIL: '/notice/:noticeId'
FAQ: '/faq'
QNA: '/qna'
QNA_DETAIL: '/qna/:qnaId'
ABOUT: '/about'               # OpenAPI 소개
TERMS: '/terms'               # 이용약관
PRIVACY: '/privacy'           # 개인정보 처리방침
LOGIN: '/login'
REGISTER: '/register'
```

2. USER - 일반 사용자 페이지 (User 인증 필요)

```
DASHBOARD: '/dashbd'
PROFILE: '/profile'
QNA_CREATE: '/user/qna/create'
QNA_HISTORY: '/user/qna/history'
OPEN_API_MANAGEMENT: '/user/openapi'
```

3. ADMIN - 관리자 페이지 (Admin 인증 필요)

기능	라우트 패턴	접근 권한
대시보드	/admin/dashbd	모든 관리자
FAQ 관리	/admin/faqs/*	VIEWER+ (조회), EDITOR+ (편집)
QnA 관리	/admin/qnas/*	VIEWER+ (조회), EDITOR+ (편집)
공지사항 관리	/admin/notices/*	VIEWER+ (조회), EDITOR+ (편집)
사용자 관리	/admin/users/*	VIEWER+ (조회), ADMIN+ (편집)
운영자 관리	/admin/operators/*	S-ADMIN 전용
OpenAPI 관리	/admin/openapi/clients/* , /admin/openapi/requests/*	모든 관리자 (조회), ADMIN+ (승인/거부)
코드 관리	/admin/code/*	S-ADMIN 전용

4. COMMON - 공통 페이지

```
NOT_FOUND: '/404'
ERROR: '/error'
```

중첩 객체 구조:

```
  ADMIN: {
    FAQ: {
      LIST: '/admin/faqs',
      CREATE: '/admin/faqs/create',
      EDIT: '/admin/faqs/:id/edit',
      DETAIL: '/admin/faqs/:id',
    },
    // ... 다른 기능들
  }
```

6.1.2 `ROUTE_META` - 네비게이션 메타데이터

메타데이터 구조:

```
  export const ROUTE_META = {
    [ROUTES.ADMIN.DASHBOARD]: {
      title: '대시보드',
      icon: 'Dashboard',
      requiresAuth: true,
      minRole: CODE_SYS_ADMIN_ROLES.VIEWER,
    },

    [ROUTES.ADMIN.OPERATORS.LIST]: {
      title: '운영자 관리',
      icon: 'AdminPanelSettings',
      requiresAuth: true,
      minRole: CODE_SYS_ADMIN_ROLES.SUPER_ADMIN,  // S-ADMIN 전용
    },
  }
```

활용:

- `title` : 페이지 제목 (네비게이션, 브레드크럼)
- `icon` : Material-UI 아이콘 이름
- `requiresAuth` : 인증 필요 여부
- `minRole` : 최소 요구 권한

6.1.3 `RouteUtils`

헬퍼 함수:

```
  export const RouteUtils = {
    // 동적 라우트 생성 (예: /faq/:id → /faq/123)
    createDynamicRoute: (route: string, params: Record<string, string | number>): string => {
      let result = route;
      Object.entries(params).forEach(([key, value]) => {
        result = result.replace(`:`, String(value));
      });
      return result;
    },

    // FAQ 상세 페이지
    createFaqDetailRoute: (id: string | number): string => {
      return RouteUtils.createDynamicRoute(ROUTES.USER.FAQ_DETAIL, { id });
    },

    // 관리자 FAQ 편집
    createAdminFaqEditRoute: (id: string | number): string => {
      return RouteUtils.createDynamicRoute(ROUTES.ADMIN.FAQ.EDIT, { id });
    },
  };
```

사용 예시:

```
// FAQ 상세 페이지로 이동
navigate(RouteUtils.createFaqDetailRoute(123));  // /user/faq/123

// 관리자 FAQ 편집 페이지로 이동
navigate(RouteUtils.createAdminFaqEditRoute(456));  // /admin/faqs/456/edit
```

6.2 라우팅 Flow

6.2.1 URL 접근 시 권한 체크 Flow

파일: `src/App.tsx`

```
<Routes>
  <Route path="/" element={<Layout />}>
    {/* 공개 페이지 (로그인 불필요) */}
    <Route index element={<Home />} />
    <Route path="/faq" element={<FaqList />} />
    <Route path="/login" element={<Login />} />

    {/* 일반 사용자 페이지 (PrivateRoute Guard) */}
    <Route
      path="/dashbd"
      element={
        <PrivateRoute>
          <Dashboard />
        </PrivateRoute>
      }
    />

    {/* 관리자 페이지 (AdminProtectedRoute Guard) */}
    <Route
      path="/admin/faqs"
      element={
        <AdminProtectedRoute>
          <AdminFaqList />
        </AdminProtectedRoute>
      }
    />
  </Route>
</Routes>
```

권한 체크 Flow:

```
사용자가 URL 접근 (예: /admin/faqs)
    │
    ▼
React Router - Route 매칭
    │
    ▼
<AdminProtectedRoute> Guard
    │
    ├─ isAdminAuthenticated() 호출
    │   │
    │   ├─ Admin 토큰 유효?
    │   │   └─ YES → 페이지 렌더링
    │   │
    │   └─ Admin 토큰 없음/만료?
    │       └─ NO → /admin/login 리다이렉트
    │           (state: { from: '/admin/faqs' } 저장)
    │
    ▼
페이지 컴포넌트 (AdminFaqList)
    │
    ▼
UI 권한 체크 (hasContentEditPermission)
    │
    ├─ VIEWER → 조회만 가능 (버튼 비활성화)
    └─ EDITOR+ → 모든 기능 가능
```

6.2.2 인증 실패 시 리다이렉트

PrivateRoute 리다이렉트:

```tsx
// src/components/ProtectedRoute.tsx
if (!isUserAuthenticated()) {
  return <Navigate to={ROUTES.PUBLIC.LOGIN} state={{ from: location }} replace />;
}
```

AdminProtectedRoute 리다이렉트:

```tsx
if (!isAdminAuthenticated()) {
  return <Navigate to={ROUTES.ADMIN.LOGIN} state={{ from: location }} replace />;
}
```

로그인 후 복원:

```tsx
// src/pages/user/Login.tsx
const location = useLocation();
const from = location.state?.from?.pathname || '/dashbd';

const handleLoginSuccess = () => {
  navigate(from, { replace: true });  // 원래 페이지로 복원
};
```

시나리오:

1. 사용자가 /user/qna/create 접근 (비로그인)
2. PrivateRoute에서 인증 실패 감지
3. /login으로 리다이렉트 (state: { from: '/user/qna/create' })
4. 사용자 로그인 성공
5. /user/qna/create로 자동 복원

6.2.3 권한 부족 시 처리

UI 레벨 권한 체크:

```
// src/pages/admin/FaqList.tsx
const adminRole = getAdminRole();
const canEdit = hasContentEditPermission(adminRole);

return (
  <div>
    <DataTable data={faqs} />

    {canEdit ? (
      <Button onClick={handleCreate}>생성</Button>
    ) : (
      <Tooltip title="편집 권한이 없습니다.">
        <span>
          <Button disabled>생성</Button>
        </span>
      </Tooltip>
    )}
  </div>
);
```

권한 부족 시 처리 방식:

1. **버튼 비활성화**: VIEWER는 생성/수정/삭제 버튼 비활성화
2. **툴팁 표시**: "편집 권한이 없습니다." 메시지
3. **메뉴 숨김**: S-ADMIN 전용 메뉴는 다른 역할에게 숨김
4. **경고 메시지**: 직접 URL 접근 시 "접근 권한이 없습니다." 표시

> **중요**: Frontend의 권한 체크는 UX 개선 목적이며, Backend에서 실제 권한 검증을 수행합니다.

7. 주요 페이지 상세

> **페이지별 설명 템플릿**:
>
> - 페이지 목적
> - URL 경로
> - 접근 권한 (Public / User / Admin 역할별)
> - 주요 기능
> - 사용 API (API 규격서 참조)
> - 권한별 UI 차이 (VIEWER/EDITOR/ADMIN/S-ADMIN)
> - 예외 처리 (인증 실패, 권한 부족, API 오류)

7.1 공개 페이지 (Public) - 인증 불필요

7.1.1 홈 페이지 - `/`

페이지: 서비스 메인 페이지 (공개 + 선택적 인증)

URL 경로: `/`

파일: `src/pages/user/Home.tsx`

접근 권한:

- **Public** (로그인 불필요)
- 로그인 시 User 정보 표시 (선택적 인증)

주요 기능:

- 서비스 소개
- 최근 공지사항 미리보기
- FAQ 미리보기
- 로그인 시 대시보드 바로가기

API:

- `publicApiFetch` 사용 (토큰 선택적 포함)

Error:

- API 오류 → 에러 메시지 표시 (서비스 계속 이용 가능)

7.1.2 로그인 페이지 - `/login`

페이지: 일반 사용자 로그인

URL 경로: `/login`

파일: `src/pages/user/Login.tsx`

접근 권한:

- **Public** (로그인 없이 접근 가능)
- 이미 로그인된 경우 홈 또는 원래 페이지로 리다이렉트

주요 기능:

- 이메일/비밀번호 입력
- 로그인 처리 (`loginUser` API)
- 로그인 실패 시 에러 메시지 표시
- 로그인 성공 시 원래 페이지 복원 (`location.state.from`)
- Admin 로그인 정리 (User 우선권 확보)

API (상세는 API 규격서 참조):

- `POST /api/user/login` - 사용자 로그인

주요 처리 Flow:

1. 이미 User 로그인? → 홈 또는 원래 페이지로 리다이렉트
2. 이메일/비밀번호 입력
3. Admin 세션 정리 (clearLoginInfoByType('A'))
4. loginUser() API 호출
5. 성공 시 → location.state.from 또는 '/dashbd'로 이동
6. 실패 시 → 에러 메시지 표시

Error:

- 로그인 실패 (`LOGIN_FAILED`) → "아이디와 비밀번호를 확인해주세요."
- 네트워크 오류 → "네트워크 오류가 발생했습니다."

7.1.3 회원가입 페이지 - `/register`

페이지: 일반 사용자 회원가입

URL 경로: `/register`

파일: `src/pages/user/Register.tsx`

접근 권한:

- **Public** (로그인 없이 접근 가능)

주요 기능:

- 이메일, 비밀번호, 이름 입력
- Common 패키지 검증 함수 사용
  - `isValidEmail(email)`
  - `isValidPassword(password)`
  - `isValidName(name)`
- 회원가입 처리
- 성공 시 로그인 페이지로 이동

API (상세는 API 규격서 참조):

- `POST /api/user/register` - 사용자 회원가입

Error:

- 이메일 중복 → "이미 사용 중인 이메일입니다."
- 비밀번호 검증 실패 → "8자 이상, 영문/숫자/특수문자 포함"
- 네트워크 오류 → 에러 메시지 표시

7.1.4 FAQ 목록 (공개) - `/faq`

페이지: 자주 묻는 질문 목록 조회 (공개)

URL 경로: `/faq`

파일: `src/pages/user/FaqList.tsx`

접근 권한:

- **Public** (로그인 불필요)

주요 기능:

- FAQ 목록 조회 (페이징, 검색)
- 카테고리별 필터링
- FAQ 상세 보기

API (상세는 API 규격서 참조):

- `GET /api/public/faq` - FAQ 목록 조회

Error:

- API 오류 → 에러 메시지 표시
- 빈 목록 → "등록된 FAQ가 없습니다." 표시

7.1.5 QnA 목록/상세 (공개) - `/qna`, `/qna/:qnaId`

페이지: 질문과 답변 목록 및 상세 조회 (공개)

URL 경로:

- 목록: `/qna`
- 상세: `/qna/:qnaId`

파일:

- `src/pages/user/QnaList.tsx`
- `src/pages/user/QnaDetail.tsx`

접근 권한:

- **Public** (로그인 불필요)

주요 기능:

- QnA 목록 조회 (페이징, 검색)
- 답변 완료/대기 상태별 필터링
- QnA 상세 보기 (질문 + 답변)

API (상세는 API 규격서 참조):

- `GET /api/public/qna` - QnA 목록 조회
- `GET /api/public/qna/:id` - QnA 상세 조회

Error:

- QnA 없음 (404) → "질문을 찾을 수 없습니다."
- API 오류 → 에러 메시지 표시

7.1.6 공지사항 목록/상세 (공개) - `/notice`, `/notice/:noticeId`

페이지: 공지사항 목록 및 상세 조회 (공개)

URL 경로:

- 목록: `/notice`
- 상세: `/notice/:noticeId`

파일:

- `src/pages/user/NoticeList.tsx`
- `src/pages/user/NoticeDetail.tsx`

접근 권한:

- **Public** (로그인 불필요)

주요 기능:

- 공지사항 목록 조회 (페이징)
- 공지사항 상세 보기

API (상세는 API 규격서 참조):

- `GET /api/public/notice` - 공지사항 목록 조회
- `GET /api/public/notice/:id` - 공지사항 상세 조회

Error:

- 공지사항 없음 (404) → "공지사항을 찾을 수 없습니다."
- API 오류 → 에러 메시지 표시

7.1.7 정적 페이지 - `/about`, `/terms`, `/privacy`

페이지: 정적 콘텐츠 페이지

URL 경로:

- `/about` - OpenAPI 소개
- `/terms` - 이용약관
- `/privacy` - 개인정보 처리방침

파일:

- `src/pages/public/OpenApiAbout.tsx`
- `src/pages/public/Terms.tsx`
- `src/pages/public/Privacy.tsx`

접근 권한:

- **Public** (로그인 불필요)

주요 기능:

- 정적 콘텐츠 표시
- API 호출 없음

7.2 사용자 페이지 (User) - 일반 사용자 인증 필요

7.2.1 사용자 대시보드 - `/dashbd`

페이지: 사용자 메인 대시보드

URL 경로: `/dashbd`

파일: `src/pages/user/Dashboard.tsx`

접근 권한:

- **User 인증 필요** (`PrivateRoute`)

주요 기능:

- 사용자 정보 표시
- My QnA 통계 (작성한 QnA 수, 답변 대기 중)
- My OpenAPI 키 통계 (활성 키, 승인 대기)
- 최근 공지사항

API (상세는 API 규격서 참조):

- `GET /api/user/stats` - 사용자 통계 (향후)
- `GET /api/user/qna` - 나의 QnA 목록
- `GET /api/user/open-api` - 나의 API 키 목록

Error:

- 인증 실패 → `/login` 리다이렉트
- API 오류 → 해당 섹션만 에러 표시

7.2.2 프로필 관리 - `/profile`

페이지: 사용자 프로필 조회/수정, 비밀번호 변경

URL 경로: `/profile`

파일: `src/pages/user/UserProfile.tsx`

접근 권한:

- **User 인증 필요** (`PrivateRoute`)

주요 기능:

- 프로필 조회 (이메일, 이름)
- 프로필 수정 (이름)
- 비밀번호 변경

API (상세는 API 규격서 참조):

- `GET /api/user/profile` - 프로필 조회
- `PUT /api/user/profile` - 프로필 수정
- `PUT /api/user/password` - 비밀번호 변경

Error:

- 인증 실패 → `/login` 리다이렉트
- 비밀번호 검증 실패 → "현재 비밀번호가 일치하지 않습니다."
- API 오류 → 에러 메시지 표시

7.2.3 My QnA 관리 - `/user/qna/*`

페이지: 나의 질문 작성, 조회, 관리

URL 경로:

- 생성: `/user/qna/create`
- 히스토리: `/user/qna/history`

파일:

- `src/pages/user/QnaCreate.tsx`
- `src/pages/user/QnaHistory.tsx`

접근 권한:

- **User 인증 필요** (`PrivateRoute`)

주요 기능:

- QnA 생성 (제목, 내용, 공개 여부)
- 나의 QnA 목록 조회
- QnA 상세 보기 (질문 + 관리자 답변)

API (상세는 API 규격서 참조):

- `POST /api/user/qna` - QnA 생성
- `GET /api/user/qna` - 나의 QnA 목록
- `GET /api/user/qna/:id` - QnA 상세

Error:

- 인증 실패 → `/login` 리다이렉트
- 입력 검증 실패 → "제목과 내용을 입력해주세요."

- API 오류 → 에러 메시지 표시

7.2.4 OpenAPI 키 관리 - `/user/openapi`

페이지: OpenAPI 인증키 신청, 조회, 관리

URL 경로: `/user/openapi`

파일: `src/pages/user/OpenApiManagement.tsx`

접근 권한:

- **🔒 User 인증 필요** (`PrivateRoute`)

주요 기능:

- API 키 신청 (키 이름, 설명, 유효기간)
- API 키 목록 조회
- API 키 상세 (키 정보, 유효기간, 승인 상태)
- API 키 갱신 신청 (유효기간 연장)
- API 키 삭제 (논리 삭제)
- API 키 복사 (클립보드)

API (상세는 API 규격서 참조):

- `GET /api/user/open-api` - 나의 API 키 목록
- `POST /api/user/open-api` - API 키 신청
- `PUT /api/user/open-api/:id/extend` - API 키 연장 신청
- `DELETE /api/user/open-api/:id` - API 키 삭제

API 키 상태:

- **PENDING** (`activeYn='N'` , `activeAt=null`) - 승인 대기
- **ACTIVE** (`activeYn='Y'` , 유효기간 내) - 활성
- **EXPIRED** (`activeYn='Y'` , 유효기간 만료) - 만료
- **REJECTED** (`keyRejectReason` 존재) - 거부됨

API 키 형식:

- **형식**: 60자 길이의 16진수 문자열 (hex)
- **정규식**: `/^[a-f0-9]{60}$/`
- **생성 방식**: Backend의 `authKeyGenerator.generate()` 함수 사용
  - `crypto.randomBytes(30)` 기반 생성 (30 bytes = 60 hex characters)
- **예시**: `a1b2c3d4e5f67890123456789012345678901234567890abcdef1234567890abcdef123456`
- **표시**: Frontend에서 키 복사 기능 제공 (클립보드 복사)

권한별 UI 차이:

- 모든 사용자 동일 (자신의 API 키만 관리)

Error:

- 인증 실패 → `/login` 리다이렉트
- 입력 검증 실패 → "키 이름과 설명을 입력해주세요."
- API 오류 → 에러 메시지 표시

7.2.5 공지사항 조회 - `/user/notice/*`

페이지: 사용자용 공지사항 목록/상세

URL 경로:

- 목록: `/user/notice`
- 상세: `/user/notice/:id`

파일:

- `src/pages/user/NoticeList.tsx`
- `src/pages/user/NoticeDetail.tsx`

접근 권한:

- **User 인증 필요** (`PrivateRoute`)

주요 기능:

- 공지사항 목록 조회 (페이징)
- 공지사항 상세 보기

API (상세는 API 규격서 참조):

- `GET /api/user/notice` - 공지사항 목록
- `GET /api/user/notice/:id` - 공지사항 상세

Error:

- 인증 실패 → `/login` 리다이렉트
- 공지사항 없음 (404) → "공지사항을 찾을 수 없습니다."

7.3 관리자 페이지 (Admin) - 관리자 인증 필요

7.3.1 관리자 로그인 - `/admin/login`

페이지: 관리자 로그인

URL 경로: `/admin/login`

파일: `src/pages/admin/AdminLogin.tsx`

접근 권한:

- **Public** (로그인 없이 접근 가능)
- 이미 관리자 로그인된 경우 `/admin/dashbd` 로 리다이렉트

주요 기능:

- 이메일/비밀번호 입력
- 관리자 로그인 처리 (`loginAdmin` API)
- 로그인 성공 시 원래 페이지 복원
- User 로그인 정리 (Admin 우선권 확보)

API (상세는 API 규격서 참조):

- `POST /api/admin/login` - 관리자 로그인

주요 처리 Flow:

```
1. 이미 Admin 로그인? → /admin/dashbd로 리다이렉트
2. 이메일/비밀번호 입력
3. User 세션 정리 (clearLoginInfoByType('U'))
4. loginAdmin() API 호출
5. 성공 시 → location.state.from 또는 '/admin/dashbd'로 이동
6. 실패 시 → 에러 메시지 표시
```

Error:

- 로그인 실패 → "관리자 정보를 확인해주세요."

- 네트워크 오류 → 에러 메시지 표시

7.3.2 관리자 대시보드 - `/admin/dashbd`

페이지: 관리자 메인 대시보드

URL 경로: `/admin/dashbd`

파일: `src/pages/admin/AdminDashboard.tsx`

접근 권한:

- **Admin 인증 필요** (`AdminProtectedRoute`)
- 모든 관리자 역할 접근 가능 (VIEWER+)

주요 기능:

- OpenAPI 키 통계 (총 키, 활성 키, 만료 키, 승인 대기)
- QnA 통계 (총 QnA, 답변 완료, 답변 대기)
- 최근 활동 현황

API (상세는 API 규격서 참조):

- `GET /api/admin/openapi/stats` - OpenAPI 통계
- `GET /api/admin/qna/stats` - QnA 통계

권한별 UI 차이:

- 모든 관리자 동일 (통계 조회만)

Error:

- 인증 실패 → `/admin/login` 리다이렉트
- API 오류 → "통계 데이터를 불러오는 중 오류가 발생했습니다."

7.3.3 관리자 프로필 - `/admin/profile`

페이지: 관리자 프로필 조회/수정, 비밀번호 변경

URL 경로: `/admin/profile`

파일: `src/pages/admin/AdminProfile.tsx`

접근 권한:

- **Admin 인증 필요** (`AdminProtectedRoute`)

주요 기능:

- 프로필 조회 (이메일, 이름, 역할)
- 프로필 수정 (이름)
- 비밀번호 변경

API (상세는 API 규격서 참조):

- `GET /api/admin/profile` - 프로필 조회
- `PUT /api/admin/profile` - 프로필 수정
- `PUT /api/admin/password` - 비밀번호 변경

권한별 UI 차이:

- 모든 관리자 동일 (자신의 프로필만 수정)

Error:

- 인증 실패 → `/admin/login` 리다이렉트
- 비밀번호 검증 실패 → "현재 비밀번호가 일치하지 않습니다."

7.3.4 FAQ 관리 (EDITOR+) - `/admin/faqs/*`

페이지: FAQ 생성, 수정, 삭제, 관리

URL 경로:

- 목록: `/admin/faqs`
- 생성: `/admin/faqs/create`
- 상세: `/admin/faqs/:id`
- 수정: `/admin/faqs/:id/edit`

파일:

- `src/pages/admin/FaqList.tsx`
- `src/pages/admin/FaqCreate.tsx`
- `src/pages/admin/FaqDetail.tsx`
- `src/pages/admin/FaqEdit.tsx`

접근 권한:

- **Admin 인증 필요** (`AdminProtectedRoute`)
- **조회**: VIEWER+ (모든 관리자)
- **생성/수정/삭제**: EDITOR+ (`hasContentEditPermission`)

주요 기능:

- FAQ 목록 조회 (페이징, 검색, 카테고리 필터)
- FAQ 생성 (제목, 내용, 카테고리)
- FAQ 수정
- FAQ 삭제 (논리 삭제)
- FAQ 일괄 삭제

API (상세는 API 규격서 참조):

- `GET /api/admin/faq` - FAQ 목록
- `POST /api/admin/faq` - FAQ 생성
- `GET /api/admin/faq/:id` - FAQ 상세
- `PUT /api/admin/faq/:id` - FAQ 수정
- `DELETE /api/admin/faq/:id` - FAQ 삭제
- `DELETE /api/admin/faq/batch` - FAQ 일괄 삭제

권한별 UI 차이:

- **VIEWER**: 목록/상세 조회만, 생성/수정/삭제 버튼 숨김
- **EDITOR/ADMIN/S-ADMIN**: 모든 기능 사용 가능

Error:

- 인증 실패 → `/admin/login` 리다이렉트
- 권한 부족 → 버튼 비활성화 + "편집 권한이 없습니다." 툴팁
- 입력 검증 실패 → "제목과 내용을 입력해주세요."
- API 오류 → 에러 메시지 표시

7.3.5 QnA 관리 (EDITOR+) - `/admin/qnas/*`

페이지: QnA 조회, 답변 작성, 관리

URL 경로:

- 목록: `/admin/qnas`
- 상세: `/admin/qnas/:id`
- 답변: `/admin/qnas/:id/reply`
- 수정: `/admin/qnas/:id/edit`

파일:

- `src/pages/admin/QnaManage.tsx`
- `src/pages/admin/QnaDetail.tsx`
- `src/pages/admin/QnaReply.tsx`
- `src/pages/admin/QnaEdit.tsx`

접근 권한:

- **Admin 인증 필요** (`AdminProtectedRoute`)
- **조회**: VIEWER+ (모든 관리자)
- **답변/수정/삭제**: EDITOR+ (`hasContentEditPermission`)

주요 기능:

- QnA 목록 조회 (페이징, 검색, 답변 상태 필터)
- QnA 상세 보기 (질문 + 답변)
- QnA 답변 작성
- QnA 수정
- QnA 삭제 (논리 삭제)
- QnA 일괄 삭제

API (상세는 API 규격서 참조):

- `GET /api/admin/qna` - QnA 목록
- `GET /api/admin/qna/:id` - QnA 상세
- `POST /api/admin/qna/:id/reply` - 답변 작성
- `PUT /api/admin/qna/:id` - QnA 수정
- `DELETE /api/admin/qna/:id` - QnA 삭제
- `DELETE /api/admin/qna/batch` - QnA 일괄 삭제

권한별 UI 차이:

- **VIEWER**: 목록/상세 조회만
- **EDITOR/ADMIN/S-ADMIN**: 답변 작성, 수정, 삭제 가능

Error:

- 인증 실패 → `/admin/login` 리다이렉트
- 권한 부족 → 버튼 비활성화
- API 오류 → 에러 메시지 표시

7.3.6 공지사항 관리 (EDITOR+) - `/admin/notices/*`

페이지: 공지사항 생성, 수정, 삭제, 관리

URL 경로:

- 목록: `/admin/notices`
- 생성: `/admin/notices/create`
- 상세: `/admin/notices/:id`
- 수정: `/admin/notices/:id/edit`

파일:

- `src/pages/admin/NoticeManage.tsx`
- `src/pages/admin/NoticeCreate.tsx`
- `src/pages/admin/NoticeDetail.tsx`
- `src/pages/admin/NoticeEdit.tsx`

접근 권한:

- **Admin 인증 필요** (`AdminProtectedRoute`)
- **조회**: VIEWER+ (모든 관리자)

- **생성/수정/삭제**: EDITOR+ (`hasContentEditPermission`)

주요 기능:

- 공지사항 목록 조회 (페이징, 검색)
- 공지사항 생성 (제목, 내용)
- 공지사항 수정
- 공지사항 삭제 (논리 삭제)
- 공지사항 일괄 삭제

API (상세는 API 규격서 참조):

- `GET /api/admin/notice` - 공지사항 목록
- `POST /api/admin/notice` - 공지사항 생성
- `GET /api/admin/notice/:id` - 공지사항 상세
- `PUT /api/admin/notice/:id` - 공지사항 수정
- `DELETE /api/admin/notice/:id` - 공지사항 삭제
- `DELETE /api/admin/notice/batch` - 공지사항 일괄 삭제

권한별 UI 차이:

- **VIEWER**: 목록/상세 조회만
- **EDITOR/ADMIN/S-ADMIN**: 모든 기능 사용 가능

Error:

- 인증 실패 → `/admin/login` 리다이렉트
- 권한 부족 → 버튼 비활성화
- API 오류 → 에러 메시지 표시

7.3.7 사용자 관리 (ADMIN+) - `/admin/users/*`

페이지: 일반 사용자 계정 조회, 생성, 수정, 삭제

URL 경로:

- 목록: `/admin/users`
- 생성: `/admin/users/create`
- 상세: `/admin/users/:id`
- 수정: `/admin/users/:id/edit`

파일:

- `src/pages/admin/UserManagement.tsx`
- `src/pages/admin/UserCreate.tsx`
- `src/pages/admin/UserDetail.tsx`
- `src/pages/admin/UserEdit.tsx`

접근 권한:

- **Admin 인증 필요** (`AdminProtectedRoute`)
- **조회**: VIEWER+ (`hasUserAccountReadPermission`)
- **생성/수정/삭제**: ADMIN+ (`hasUserAccountEditPermission`)

주요 기능:

- 사용자 목록 조회 (페이징, 검색, 상태 필터)
- 사용자 생성
- 사용자 상세 보기 (가입일, 최근 접속일, 보유 API 키 등)
- 사용자 수정 (이름, 상태)
- 사용자 삭제 (논리 삭제)
- 사용자 일괄 삭제
- 사용자 비밀번호 초기화

API (상세는 API 규격서 참조):

- `GET /api/admin/user-accounts` - 사용자 목록
- `POST /api/admin/user-accounts` - 사용자 생성
- `GET /api/admin/user-accounts/:id` - 사용자 상세
- `PUT /api/admin/user-accounts/:id` - 사용자 수정
- `DELETE /api/admin/user-accounts/:id` - 사용자 삭제
- `DELETE /api/admin/user-accounts/batch` - 사용자 일괄 삭제

권한별 UI 차이:

- **VIEWER/EDITOR**: 목록/상세 조회만, 생성/수정/삭제 버튼 숨김
- **ADMIN/S-ADMIN**: 모든 기능 사용 가능

Error:

- 인증 실패 → `/admin/login` 리다이렉트
- 권한 부족 → 버튼 비활성화 + "편집 권한이 없습니다." 툴팁
- 이메일 중복 → "이미 사용 중인 이메일입니다."
- API 오류 → 에러 메시지 표시

7.3.8 관리자 계정 관리 (S-ADMIN) - `/admin/operators/*`

페이지: 운영자(관리자) 계정 생성, 수정, 삭제, 역할 관리

URL 경로:

- 목록: `/admin/operators`
- 생성: `/admin/operators/create`
- 상세: `/admin/operators/:id`
- 수정: `/admin/operators/:id/edit`

파일:

- `src/pages/admin/OperatorManagement.tsx`
- `src/pages/admin/OperatorCreate.tsx`
- `src/pages/admin/OperatorDetail.tsx`
- `src/pages/admin/OperatorEdit.tsx`

접근 권한:

- **Admin 인증 필요** (`AdminProtectedRoute`)
- **S-ADMIN 전용** (`hasAccountManagementPermission`)

주요 기능:

- 운영자 목록 조회 (페이징, 검색, 역할 필터)
- 운영자 생성 (이메일, 비밀번호, 이름, 역할)
- 운영자 상세 보기
- 운영자 수정 (이름, 역할, 상태)
- 운영자 삭제 (논리 삭제)
- 운영자 일괄 삭제
- 이메일 중복 확인
- 비밀번호 변경

API (상세는 API 규격서 참조):

- `GET /api/admin/account` - 운영자 목록
- `POST /api/admin/account` - 운영자 생성
- `GET /api/admin/account/:id` - 운영자 상세
- `PUT /api/admin/account/:id` - 운영자 수정
- `DELETE /api/admin/account/:id` - 운영자 삭제
- `DELETE /api/admin/account/batch` - 운영자 일괄 삭제

- `POST /api/admin/account/check-email` - 이메일 중복 확인

권한별 UI 차이:

- **VIEWER/EDITOR/ADMIN**: 메뉴 자체가 숨김
- **S-ADMIN**: 모든 기능 사용 가능

Error:

- 인증 실패 → `/admin/login` 리다이렉트
- 권한 부족 (S-ADMIN 아님) → "접근 권한이 없습니다."
- 이메일 중복 → "이미 사용 중인 이메일입니다."
- API 오류 → 에러 메시지 표시

7.3.9 OpenAPI 클라이언트 관리 (전체) - `/admin/openapi/*`

페이지: OpenAPI 클라이언트 및 API 키 승인 관리

URL 경로:

- 클라이언트 목록: `/admin/openapi/clients`
- 클라이언트 상세: `/admin/openapi/clients/:id`
- 클라이언트 수정: `/admin/openapi/clients/:id/edit`
- 승인 요청 목록: `/admin/openapi/requests`
- 승인 요청 상세: `/admin/openapi/requests/:id`

파일:

- `src/pages/admin/OpenApiManage.tsx` (클라이언트 목록)
- `src/pages/admin/OpenApiDetail.tsx` (클라이언트 상세)
- `src/pages/admin/OpenApiEdit.tsx` (클라이언트 수정)
- `src/pages/admin/OpenApiRequests.tsx` (승인 요청 목록)
- `src/pages/admin/OpenApiRequestDetail.tsx` (승인 요청 상세)

접근 권한:

- **Admin 인증 필요** (`AdminProtectedRoute`)
- **조회**: VIEWER+ (모든 관리자)
- **승인/거부/연장/삭제**: ADMIN+ (`hasUserAccountEditPermission`)

주요 기능:

- OpenAPI 클라이언트 목록 조회 (사용자별 API 키 현황)
- 클라이언트 상세 보기 (보유 API 키 목록)
- API 키 승인 요청 목록 (`activeYn='N'` , `activeAt=null`)
- API 키 승인/거부
- API 키 유효기간 연장
- API 키 삭제

API (상세는 API 규격서 참조):

- `GET /api/admin/open-api` - OpenAPI 클라이언트 목록
- `GET /api/admin/open-api/:id` - 클라이언트 상세
- `PUT /api/admin/open-api/:id/approve` - API 키 승인
- `PUT /api/admin/open-api/:id/reject` - API 키 거부
- `PUT /api/admin/open-api/:id/extend` - API 키 연장
- `DELETE /api/admin/open-api/:id` - API 키 삭제

API 키 형식:

- **형식**: 60자 길이의 16진수 문자열 (hex)
- **정규식**: `/^[a-f0-9]{60}$/`
- **생성 방식**: Backend의 `authKeyGenerator.generate()` 함수 사용
  - `crypto.randomBytes(30)` 기반 생성 (30 bytes = 60 hex characters)

- **예시**: `a1b2c3d4e5f67890123456789012345678901234567890abcdef1234567890abcdef123456`

권한별 UI 차이:

- **VIEWER/EDITOR**: 목록/상세 조회만, 승인/거부/연장 버튼 숨김
- **ADMIN/S-ADMIN**: 모든 기능 사용 가능

Error:

- 인증 실패 → `/admin/login` 리다이렉트
- 권한 부족 → 버튼 비활성화
- API 오류 → 에러 메시지 표시

7.3.10 코드 관리 (S-ADMIN) - `/admin/code/*`

페이지: 공통 코드 그룹 및 코드 관리

URL 경로:

- 코드 목록: `/admin/code`
- 코드 그룹 상세: `/admin/code/group/:groupId`
- 코드 생성: `/admin/code/create`
- 코드 상세: `/admin/code/:id`
- 코드 수정: `/admin/code/:id/edit`

파일:

- `src/pages/admin/CodeManagement.tsx`
- `src/pages/admin/CodeGroupDetail.tsx`
- `src/pages/admin/CodeCreate.tsx`
- `src/pages/admin/CodeDetail.tsx`

접근 권한:

- **Admin 인증 필요** (`AdminProtectedRoute`)
- **S-ADMIN 전용** (`hasAccountManagementPermission`)

주요 기능:

- 코드 그룹 목록 조회
- 코드 그룹별 상세 (하위 코드 목록)
- 코드 생성
- 코드 수정
- 코드 삭제

API (상세는 API 규격서 참조):

- `GET /api/admin/common-code` - 코드 그룹 목록
- `GET /api/admin/common-code/group/:id` - 코드 그룹 상세
- `POST /api/admin/common-code` - 코드 생성
- `PUT /api/admin/common-code/:id` - 코드 수정
- `DELETE /api/admin/common-code/:id` - 코드 삭제

권한별 UI 차이:

- **VIEWER/EDITOR/ADMIN**: 메뉴 자체가 숨김
- **S-ADMIN**: 모든 기능 사용 가능

Error:

- 인증 실패 → `/admin/login` 리다이렉트
- 권한 부족 → "접근 권한이 없습니다."
- API 오류 → 에러 메시지 표시

7.4 공통 페이지 (Common)

7.4.1 404 페이지 - `/404`

페이지: 페이지를 찾을 수 없을 때 표시

URL 경로: `/404` (자동 리다이렉트)

접근 권한:

- **Public**

주요 기능:

- "페이지를 찾을 수 없습니다." 메시지
- 홈으로 돌아가기 버튼

7.4.2 에러 페이지 - `/error`

페이지: 시스템 오류 발생 시 표시

URL 경로: `/error`

접근 권한:

- **Public**

주요 기능:

- "시스템 오류가 발생했습니다." 메시지
- 홈으로 돌아가기 버튼

8. 공통 컴포넌트

8.1 컴포넌트 분류 및 역할

Frontend의 공통 컴포넌트는 **재사용성**과 **일관된 UI**를 위해 체계적으로 분류됩니다.

분류 체계:

1. **레이아웃 컴포넌트**: 페이지 전체 구조 (Layout, AppBar, Footer 등)
2. **관리자 전용 컴포넌트**: 관리자 페이지 전용 (SideNav, AdminPageHeader)
3. **공통 UI 컴포넌트**: 범용 재사용 컴포넌트 (DataTable, Pagination, StatusChip 등)
4. **폼 컴포넌트**: 입력 폼 (LoginForm, ProfileForm)
5. **피드백 컴포넌트**: 로딩, 에러, 토스트 등

8.2 컴포넌트 목록 (테이블 형태)

컴포넌트명	경로	설명	주요 사용처
레이아웃			
`Layout.tsx`	`/components/`	전체 레이아웃 (Header + Content + Footer)	모든 페이지
`AppBar.tsx`	`/components/`	사용자 상단 앱바	User 페이지
`AppBarCommon.tsx`	`/components/`	공통 앱바 로직	AppBar 공유
`AdminMenuBar.tsx`	`/components/`	관리자 메뉴바	Admin 페이지
`Footer.tsx`	`/components/`	푸터	모든 페이지
관리자 전용			
`AdminPageHeader.tsx`	`/components/admin/`	관리자 페이지 헤더	Admin 페이지 상단
`SideNav.tsx`	`/components/admin/`	관리자 사이드 네비게이션 (권한별 메뉴 표시)	Admin 페이지 좌측
데이터 표시			
`DataTable.tsx`	`/components/common/`	데이터 테이블 (정렬, 클릭 이벤트)	모든 목록 페이지
`TableListBody.tsx`	`/components/common/`	테이블 리스트 본문	테이블 형식 목록
`CardListBody.tsx`	`/components/common/`	카드 리스트 본문	카드 형식 목록
`ListItemCard.tsx`	`/components/common/`	리스트 아이템 카드	카드 리스트
`EmptyState.tsx`	`/components/common/`	빈 상태 표시 ("데이터가 없습니다.")	빈 목록 페이지
페이지 구성			
`PageHeader.tsx`	`/components/common/`	페이지 헤더	페이지 상단
`PageTitle.tsx`	`/components/common/`	페이지 제목	페이지 타이틀
`ListHeader.tsx`	`/components/common/`	리스트 헤더 (검색, 필터)	목록 페이지 상단
`ListScaffold.tsx`	`/components/common/`	리스트 스캐폴드 (목록 구조 템플릿)	목록 페이지
`ListTotal.tsx`	`/components/common/`	리스트 총 개수 표시	목록 페이지
`Pagination.tsx`	`/components/common/`	페이지네이션	목록 페이지 하단
입력 및 액션			
`SelectField.tsx`	`/components/common/`	셀렉트 필드 (드롭다운)	검색 필터, 폼 입력
`ThemedButton.tsx`	`/components/common/`	테마 버튼 (User/Admin 색상)	모든 버튼
`ByteLimitHelper.tsx`	`/components/common/`	바이트 제한 헬퍼 (텍스트 입력)	폼 입력

컴포넌트명	경로	설명	주요 사용처
피드백			
StatusChip.tsx	/components/common/	상태 칩 (활성/비활성/대기 등)	상태 표시
QnaTypeChip.tsx	/components/common/	QnA 타입 칩 (공개/비공개)	QnA 목록
ThemedCard.tsx	/components/common/	테마 카드	대시보드, 통계
LoadingSpinner.tsx	/components/	로딩 스피너	API 호출 중
ErrorAlert.tsx	/components/	에러 알림	에러 발생 시
다이얼로그			
CommonDialog.tsx	/components/	공통 다이얼로그 (확인/취소)	삭제 확인 등
CommonToast.tsx	/components/	공통 토스트 (알림)	성공/실패 메시지
ToastProvider.tsx	/components/	토스트 프로바이더	전역 토스트 관리
ExtendKeyDialog.tsx	/components/common/	API 키 연장 다이얼로그	OpenAPI 키 연장
폼			
LoginForm.tsx	/components/	로그인 폼 (공통)	User/Admin 로그인
ProfileForm.tsx	/components/	프로필 폼 (공통)	User/Admin 프로필
라우트 가드			
ProtectedRoute.tsx	/components/	권한 체크 Guard (PrivateRoute, AdminProtectedRoute)	보호된 라우트

총 30개 이상의 컴포넌트가 체계적으로 분류되어 재사용됩니다.

9. 유틸리티 함수

9.1 유틸리티 함수 목록 (테이블 형태)

파일	주요 함수	설명	사용처
auth.ts	isSAdmin , isAdmin , hasContentEditPermission , hasUserAccountEditPermission , hasAccountManagementPermission , hasMenuAccess , hasActionPermission	권한 체크 함수 (8개)	모든 관리자 페이지 (버튼, 메뉴 제어)
jwt.ts	isValidTokenFormat , extractTokenInfo , isTokenExpired , getTokenTimeRemaining , shouldRefreshToken	JWT 토큰 검증 및 만료 체크	인증 관련 모든 로직
apiResponseHandler.ts	enhanceApiResponse , handleApiResponse , shouldShowPopup , shouldAutoLogout , getRedirectUrl	API 응답 강화 및 에러 핸들링	모든 API 호출
date.ts	formatYmd , formatYmdHm , formatRelativeTime	날짜 포맷팅 (YYYY-MM-DD, YYYY-MM-DD HH:mm 등)	목록, 상세 페이지 (날짜 표시)
openApiStatus.ts	getOpenApiKeyStatus	OpenAPI 키 상태 판단 (PENDING, ACTIVE, EXPIRED, REJECTED)	OpenAPI 관리 페이지

9.2 Custom React Hooks (테이블 형태)

Hook	주요 기능	목적
useDataFetching	API 호출, 로딩/에러/빈 상태 자동 관리, 자동/수동 페칭	목록 페이지의 반복 코드 제거 (로딩, 에러, 빈 상태 통합 관리)
useQuerySync	URL 쿼리 파라미터 동기화 (page, limit, search 등)	검색/페이징 상태를 URL에 유지 (뒤로가기 지원, 공유 가능)
usePagination	페이지 상태 관리 (page, limit, total, totalPages)	페이지네이션 로직 통합
usePasswordValidation	비밀번호 실시간 검증 (8자 이상, 영문/숫자/특수문자 포함)	회원가입, 비밀번호 변경 시 즉시 피드백
useInputWithTrim	입력 값 자동 trim 처리 (앞뒤 공백 제거)	사용자 입력 정규화 (이메일, 이름 등)
useErrorHandler	에러 상태 관리 및 자동 초기화	에러 처리 통합 (자동 닫기 타이머)
useCommonCode	공통 코드 조회 및 캐싱	드롭다운, 필터에서 코드 목록 재사용

10. 환경 설정 및 빌드

10.1 환경 변수 설정

10.1.1 .env 파일 구조

파일 위치: `fe/.env` (개발), `fe/.env.production` (프로덕션)

Vite 환경 변수 규칙:

- 모든 환경 변수는 `VITE_` prefix 필수
- `import.meta.env.VITE_*` 로 접근

10.1.2 주요 환경 변수

변수명	설명	기본값	예시
`VITE_API_BASE_URL`	Backend API URL	`http://localhost:30000`	`http://api.example.com`
`VITE_API_TIMEOUT`	API 타임아웃 (ms)	`10000` (10초)	`30000` (30초)
`VITE_OPEN_API_DOC_URL`	OpenAPI 문서 URL	`http://localhost:8080/api-docs`	`http://api.example.com/docs`
`VITE_OPEN_API_SERVER_URL`	OpenAPI 서버 URL	`http://localhost:8080`	`http://api.example.com`
`VITE_BASE`	Base URL (서브 경로 배포)	`/`	`/admin`
`VITE_PORT`	개발 서버 포트	`5173`	`3000`

사용 예시 (`src/config.ts`):

```
export const API_BASE_URL = import.meta.env.VITE_API_BASE_URL || 'http://localhost:30000';
export const API_TIMEOUT = Number(import.meta.env.VITE_API_TIMEOUT) || 10000;
export const OPEN_API_DOC_URL = import.meta.env.VITE_OPEN_API_DOC_URL || 'http://localhost:8080/api-docs';
```

10.2 빌드 설정

10.2.1 Vite 플러그인 설정

```
import { defineConfig, loadEnv } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig(({ mode }) => {
  const env = loadEnv(mode, process.cwd(), '')
  return {
    plugins: [react()],
    base: env.VITE_BASE || '/',
    server: {
      port: Number(env.VITE_PORT) || 5173,
    },
    build: {
      outDir: 'dist',
    },
  }
})
```

플러그인:

- `@vitejs/plugin-react` : React Fast Refresh, JSX 변환

10.2.2 개발 서버 설정

- **포트**: `VITE_PORT` (기본 5173)
- **HMR**: Hot Module Replacement 자동 지원
- **CORS**: Backend API 프록시 필요 시 설정 가능

10.2.3 빌드 출력 설정

- **출력 디렉토리**: `dist/`
- **정적 자산**: `dist/assets/` (JS, CSS, 이미지 등)
- **빌드 정보**: `dist/build-info.json` (버전, 빌드 시각)

10.3 TypeScript 설정

Project References:

```
{
  "references": [
    { "path": "../packages/common" },      // Common 패키지 참조
    { "path": "./tsconfig.app.json" },     // App 설정
    { "path": "./tsconfig.node.json" }     // Node 설정 (Vite)
  ]
}
```

경로 매핑:

```
{
  "compilerOptions": {
    "baseUrl": ".",
    "paths": {
      "packages/common/*": ["../packages/common/*"]
    }
  }
}
```

10.4 빌드 및 배포

10.4.1 개발 환경 실행 (`npm run dev`)

```
cd fe
npm run dev
```

실행 결과:

```
VITE v5.0.8  ready in 500 ms

➜  Local:   http://localhost:5173/
➜  Network: http://192.168.x.x:5173/
➜  press h to show help
```

10.4.2 프로덕션 빌드 (`npm run build`)

```
cd fe
npm run build
```

빌드 단계:

```
1. rimraf dist          # 이전 빌드 삭제
2. tsc -b               # TypeScript 컴파일 (타입 체크)
3. vite build           # Vite 빌드 (번들링, 최적화)
4. node scripts/build-info.js  # 빌드 정보 생성
```

빌드 결과:

```
dist/
  index.html                  # 엔트리 HTML
  build-info.json             # 빌드 정보 (버전, 빌드 시각)
  assets/
    index-[hash].js           # 번들 JS
    index-[hash].css          # 번들 CSS
    logo-[hash].png           # 이미지 자산
```

10.4.3 빌드 결과물 (`/dist`)

파일 구조:

```
dist/
  index.html                  # SPA 엔트리 포인트
  build-info.json             # { version, buildDate }
  assets/
    index-[hash].js           # Main JS Bundle
    vendor-[hash].js          # Vendor Bundle (React, MUI 등)
    index-[hash].css          # Main CSS Bundle
```

빌드 정보 예시 (`build-info.json`):

```
{
  "version": "1.0.0",
  "buildDate": "2025-11-07 14:30:45.123"
}
```

10.4.4 배포 시 주의사항

1. SPA 라우팅 설정 (Nginx fallback)

Nginx 설정 예시:

```
location / {
  root /var/www/html/fe/dist;
  try_files  / /index.html;  # SPA Fallback
}
```

이유: React Router는 클라이언트 라우팅이므로, 모든 경로를 `index.html` 로 fallback 필요

2. 환경 변수 주입

프로덕션 빌드 전 `.env.production` 설정:

```
VITE_API_BASE_URL=https://api.production.com
VITE_API_TIMEOUT=30000
```

3. CORS 설정 확인

Backend API 서버에서 Frontend Origin 허용 필요:

```
// BE: src/index.ts
app.use(cors({
  origin: 'https://admin.production.com',  // Frontend URL
  credentials: true
}));
```

4. 빌드 정보 표시 (`scripts/build-info.js`)

빌드 버전 및 빌드 시각을 `dist/build-info.json` 으로 자동 생성하여 버전 추적 가능

11. 예외 처리 및 에러 핸들링

11.1 API 에러 처리

11.1.1 네트워크 오류, 타임아웃

네트워크 오류 처리:

```
// src/api/api.ts
try {
  const res = await fetch(url, { signal: controller.signal });
  // ...
} catch (e: any) {
  if (e.name === 'AbortError') {
    return enhanceApiResponse({
      success: false,
      errorMessage: '요청 시간이 초과되었습니다.',
      errorCode: ErrorCode.REQUEST_TIMEOUT
    });
  }

  return enhanceApiResponse({
    success: false,
    errorMessage: '네트워크 오류가 발생했습니다.',
    errorCode: ErrorCode.NETWORK_ERROR
  });
}
```

타임아웃 설정:

- 기본 타임아웃: **10초**
- `AbortController` 를 통한 요청 취소

사용자 메시지:

- 타임아웃: "요청 시간이 초과되었습니다."
- 네트워크 오류: "네트워크 오류가 발생했습니다."

11.1.2 인증 오류 (401) - 자동 토큰 갱신 시도

401 에러 처리 Flow:

```
API 요청 → 401 Unauthorized
    |
    ▼
Refresh Token 유효성 확인
    |
    ├─ 유효? → Refresh Token으로 새 토큰 발급
    |   |
    |   ├─ 성공 → 동일 API 재시도 (1회만)
    |   |   |
    |   |   ├─ 성공 → 정상 응답 반환
    |   |   └─ 실패 → 에러 응답 반환
    |   |
    |   └─ 실패 → 자동 로그아웃 + 로그인 페이지 리다이렉트
    |
    └─ 만료? → 자동 로그아웃 + 로그인 페이지 리다이렉트
```

자동 처리:

- `enhanceApiResponse()` 가 `autoLogout: true` 설정
- `getRedirectUrl()` 이 User/Admin 타입별 로그인 페이지 반환

- 토큰 제거 후 자동 리다이렉트

사용자 메시지:

- "인증이 필요합니다. 다시 로그인해주세요."

11.1.3 권한 오류 (403)

권한 오류 처리:

```
if (data.errorCode === ErrorCode.ACCESS_DENIED) {
  return {
    ...response,
    errorMessage: '접근 권한이 없습니다.',
    redirectTo: '/',  // 홈으로 리다이렉트
  };
}
```

사용자 메시지:

- "접근 권한이 없습니다."

처리 방식:

- 에러 메시지 표시 후 홈으로 리다이렉트

11.1.4 서버 오류 (500)

서버 오류 처리:

```
if (res.status >= 500) {
  return enhanceApiResponse({
    success: false,
    errorMessage: '서버 오류가 발생했습니다. 잠시 후 다시 시도해주세요.',
    errorCode: ErrorCode.INTERNAL_SERVER_ERROR
  });
}
```

사용자 메시지:

- "서버 오류가 발생했습니다. 잠시 후 다시 시도해주세요."

처리 방식:

- 에러 메시지 표시
- 재시도 없음 (사용자가 수동으로 재시도)

11.2 사용자 친화적 에러 메시지

11.2.1 ErrorCode 기반 메시지 생성 (`createUserFriendlyMessage`)

파일: `src/api/api.ts`

```
function createUserFriendlyMessage(data: any): string {
  if (data?.errorMessage) {
    return data.errorMessage;  // Backend에서 제공한 메시지 우선
  }

  if (data?.errorCode) {
    switch (data.errorCode) {
      case ErrorCode.UNAUTHORIZED:
        return '인증이 필요합니다. 다시 로그인해주세요.';
      case ErrorCode.TOKEN_EXPIRED:
        return '로그인 세션이 만료되었습니다. 다시 로그인해주세요.';
      case ErrorCode.INVALID_TOKEN:
        return '유효하지 않은 인증 정보입니다. 다시 로그인해주세요.';
      case ErrorCode.ACCESS_DENIED:
        return '접근 권한이 없습니다.';
      case ErrorCode.USER_NOT_FOUND:
        return '사용자를 찾을 수 없습니다.';
      case ErrorCode.LOGIN_FAILED:
        return '로그인에 실패했습니다. 아이디와 비밀번호를 확인해주세요.';
      case ErrorCode.NETWORK_ERROR:
        return '네트워크 오류가 발생했습니다.';
      case ErrorCode.REQUEST_TIMEOUT:
        return '요청 시간이 초과되었습니다.';
      default:
        return '오류가 발생했습니다. 다시 시도해주세요.';
    }
  }

  return '알 수 없는 오류가 발생했습니다.';
}
```

메시지 우선순위:

1. Backend `errorMessage` (있으면 우선 사용)
2. ErrorCode 기반 Frontend 메시지
3. 기본 메시지

11.2.2 Toast/Alert를 통한 에러 표시

ErrorAlert 컴포넌트:

```
// src/components/ErrorAlert.tsx
<Alert severity="error" onClose={onClose}>
  {error}
</Alert>
```

CommonToast 사용:

```
// src/components/ToastProvider.tsx
const { showToast } = useToast();

// 성공 메시지
showToast('FAQ가 성공적으로 생성되었습니다.', 'success');

// 에러 메시지
showToast('FAQ 생성에 실패했습니다.', 'error');
```

표시 방식:

- **Alert**: 페이지 상단에 고정 표시 (닫기 버튼 포함)
- **Toast**: 화면 하단에 자동 사라지는 알림 (3-5초)

11.3 토큰 갱신 실패 시 처리

11.3.1 자동 로그아웃 및 로그인 페이지 리다이렉트

처리 Flow:

```
토큰 갱신 실패
    │
    ▼
removeTokens() 호출 (현재 타입의 토큰 제거)
    │
    ▼
clearLoginInfo() 호출 (현재 타입의 사용자 정보 제거)
    │
    ▼
리다이렉트 URL 생성
    │
    ├─ Admin 타입? → /admin/login
    └─ User 타입? → /login
    │
    ▼
window.location.href = redirectTo (페이지 이동)
```

파일: `src/utils/apiResponseHandler.ts`

```typescript
export function handleApiResponse<T>(response: ApiResponse<T>, onSuccess, onError) {
  if (!response.success) {
    // 자동 로그아웃 처리
    if (response.autoLogout) {
      const userType = getUserType();
      removeTokensByType(userType === 'A' ? 'A' : 'U');
    }

    // 리다이렉트 처리
    if (response.redirectTo) {
      setTimeout(() => {
        window.location.href = response.redirectTo;
      }, 100);  // 에러 메시지 표시 후 이동
    }
  }
}
```

자동 로그아웃 발생 ErrorCode:

- `TOKEN_EXPIRED` (40102)
- `INVALID_TOKEN` (40103)
- `UNAUTHORIZED` (40101)
- `TOKEN_REQUIRED` (40104)

11.4 권한 부족 시 UI 처리

11.4.1 버튼 비활성화, 메뉴 숨김, 경고 메시지

버튼 비활성화 예시:

```
const canEdit = hasContentEditPermission(adminRole);

<Button
  disabled={!canEdit}
  onClick={handleCreate}
>
  생성
</Button>

{!canEdit && (
  <Tooltip title="편집 권한이 없습니다.">
    <InfoIcon />
  </Tooltip>
)}
```

메뉴 숨김 예시:

```
// src/components/admin/SideNav.tsx
const adminRole = getAdminRole();

{hasAccountManagementPermission(adminRole) && (
  <MenuItem onClick={() => navigate('/admin/operators')}>
    운영자 관리
  </MenuItem>
)}
```

경고 메시지 예시:

```
if (!hasContentEditPermission(adminRole)) {
  showToast('편집 권한이 없습니다.', 'error');
  return;
}
```

권한 체크 시점:

1. **라우트 가드**: `AdminProtectedRoute` (인증 체크)
2. **메뉴 렌더링**: 역할별 메뉴 표시/숨김
3. **버튼 클릭**: 액션 실행 전 권한 확인
4. **API 호출 전**: 클라이언트 측 사전 검증

12. 성능 최적화

12.1 코드 스플리팅 (Lazy Loading)

12.1.1 관리자 페이지 Lazy Loading (`React.lazy()` + `Suspense`)

파일: `src/App.tsx`

```
import { Suspense, lazy } from 'react';
import LoadingSpinner from './components/LoadingSpinner';

// 관리자 페이지 Lazy Loading
const AdminProfile = lazy(() => import('./pages/admin/AdminProfile'));
const AdminFaqList = lazy(() => import('./pages/admin/FaqList'));
const AdminQnaList = lazy(() => import('./pages/admin/QnaManage'));
const AdminNoticeList = lazy(() => import('./pages/admin/NoticeManage'));
const AdminOpenApiClients = lazy(() => import('./pages/admin/OpenApiManage'));
const AdminOpenApiRequests = lazy(() => import('./pages/admin/OpenApiRequests'));
// ... 추가 관리자 페이지

// Suspense로 래핑
<Route
  path="/admin/faqs"
  element={
    <AdminProtectedRoute>
      <Suspense fallback={<LoadingSpinner loading={true} />}>
        <AdminFaqList />
      </Suspense>
    </AdminProtectedRoute>
  }
/>
```

12.1.2 구현 위치: `App.tsx`

적용 페이지:

- 모든 관리자 페이지 (`/admin/*`)
- 공개 페이지 (초기 로드 필요)
- 사용자 페이지 (초기 로드 필요)

12.2 Vite 빌드 최적화

12.2.1 Tree Shaking (기본 제공)

Vite는 **ES Module** 기반으로 사용하지 않는 코드를 자동 제거합니다.

12.2.2 번들 크기 최적화

Chunk 분리:

```
dist/assets/
  index-[hash].js      # Main Bundle
  vendor-[hash].js     # Vendor Bundle (React, MUI 등)
  admin-[hash].js      # Admin 페이지 (Lazy Loading)
```

12.3 API 요청 최적화

12.3.1 토큰 자동 갱신 (중복 요청 방지)

중복 갱신 방지 메커니즘:

- 여러 API가 동시에 호출되어도 토큰 갱신은 1회만 발생
- `ensureValidToken()` 함수가 갱신 중 여부 체크
- 불필요한 중복 요청 방지

효과:

- 서버 부하 감소
- 토큰 갱신 속도 향상

12.3.2 API 타임아웃 설정 (10초)

기본 타임아웃: 10초 (`API_TIMEOUT`)

커스텀 타임아웃:

```
// 대용량 데이터 조회 시 30초
await apiFetch('/api/admin/logs', { timeoutMs: 30000 });
```

13. 보안

13.1 토큰 보안

13.1.1 LocalStorage 사용 (XSS 주의 필요)

저장 위치: `LocalStorage`

보안 고려사항:

- **XSS 공격에 취약**: LocalStorage는 JavaScript로 접근 가능
- **HttpOnly Cookie 대안**: 현재는 LocalStorage 사용 (향후 개선 검토)
- **HTTPS 필수**: 프로덕션 환경에서는 HTTPS 사용 필수

완화 방안:

- 토큰 형식 검증 (`isValidTokenFormat`)
- 토큰 만료 체크 (자동 정리)
- Content Security Policy (CSP) 설정 권장

13.1.2 토큰 형식 검증 (`isValidTokenFormat`)

```
// src/utils/jwt.ts
export function isValidTokenFormat(token: string): boolean {
  if (!token || typeof token !== 'string') return false;

  const parts = token.split('.');
  return parts.length === 3;  // header.payload.signature
}
```

검증 시점:

- 토큰 저장 시 (`saveTokens`)
- 토큰 조회 시 (`getAccessToken` , `getRefreshToken`)
- 유효하지 않은 형식 → 자동 제거

13.1.3 토큰 만료 체크 (`isTokenExpired`)

```
// src/utils/jwt.ts
export function isTokenExpired(token: string): boolean {
  try {
    const decoded = jwtDecode(token) as any;
    if (!decoded || !decoded.exp) return true;

    const currentTime = Math.floor(Date.now() / 1000);
    return decoded.exp < currentTime;
  } catch {
    return true;
  }
}
```

검증 시점:

- 인증 상태 확인 시 (`isUserAuthenticated` , `isAdminAuthenticated`)
- API 요청 전 (`ensureValidToken`)
- 라우트 가드 실행 시 (`PrivateRoute` , `AdminProtectedRoute`)

만료된 토큰 처리:

- 자동 제거 (`validateAndCleanTokens`)
- Refresh Token으로 갱신 시도

- 갱신 실패 시 로그인 페이지 리다이렉트

13.2 권한 체크

13.2.1 Frontend 권한 체크 (UI 제어 목적)

파일: `src/utils/auth.ts`

목적: **사용자 경험(UX) 개선**

- 권한 없는 버튼 비활성화
- 권한 없는 메뉴 숨김
- 불필요한 API 호출 방지

13.2.2 Backend 권한 체크 (실제 보안 담당)

실제 보안: Backend의 `authMiddleware`, `adminAuthMiddleware`에서 담당

이중 검증 필요:

```
Frontend 권한 체크 (UX)
    │
    ▼
API 요청
    │
    ▼
Backend 권한 체크 (보안)  실제 보안
    │
    ├─ 권한 있음 → 처리
    └─ 권한 없음 → 403 Forbidden
```

13.2.3 이중 검증 필요성 (Frontend는 UX, Backend는 보안)

Frontend 권한 체크:

- **UX 개선**: 권한 없는 버튼 숨김
- **사전 검증**: 불필요한 API 호출 방지
- **보안 역할 아님**: 우회 가능

Backend 권한 체크:

- **실제 보안**: 서버에서 강제 검증
- **우회 불가능**: 클라이언트에서 조작 불가
- **최종 검증**: 모든 요청에 대해 검증

권장 사항:

- Frontend: UX 개선 목적으로만 사용
- Backend: 실제 보안 검증 (필수)
- 두 검증 모두 구현하여 **UX와 보안 모두 확보**

14. 부록

14.1 주요 npm 패키지 설명 (테이블 형태)

패키지	버전	카테고리	설명
`react`	^18.2.0	Core	UI 라이브러리
`react-dom`	^18.2.0	Core	React DOM 렌더링
`react-router-dom`	^6.20.1	Routing	SPA 라우팅
`typescript`	^5.x	Dev	정적 타입 체킹
`@mui/material`	^5.15.0	UI	Material-UI 컴포넌트
`@mui/icons-material`	^5.15.0	UI	Material-UI 아이콘
`@emotion/react`	^11.11.1	UI	CSS-in-JS (MUI 의존성)
`@emotion/styled`	^11.11.0	UI	Styled Components
`jwt-decode`	^4.0.0	Auth	JWT 토큰 디코딩
`axios`	^1.11.0	HTTP	HTTP 클라이언트 (실제로는 fetch 사용)
`vite`	^5.0.8	Build	빌드 도구 및 개발 서버
`@vitejs/plugin-react`	^4.2.1	Build	Vite React 플러그인
`@iitp-dabt/common`	file:../packages/common	Shared	BE/FE 공유 코드 (검증, ErrorCode 등)
`rimraf`	-	Dev	디렉토리 삭제 (빌드 전 정리)

14.2 공통 타입 정의

파일 목록:

파일	설명
`types/api.ts`	API 요청/응답 타입, `ApiResponse<T>` 인터페이스
`types/errorCodes.ts`	ErrorCode 타입 (Common 패키지 재정의)

주요 타입:

`ApiResponse<T>`:

```typescript
export interface ApiResponse<T = any> {
  success: boolean;
  data?: T;
  errorCode?: number;
  errorMessage?: string;
  showPopup?: boolean;
  redirectTo?: string;
  autoLogout?: boolean;
  details?: any;
}
```

`DataState<T>`:

```
export type DataState<T> =
  | { status: 'loading' }
  | { status: 'success'; data: T }
  | { status: 'empty' }
  | { status: 'error'; error: string };
```

14.3 프로젝트 아키텍처 가이드 참조

관련 섹션:

- **3. 권한 체계 및 접근 제어**: User/Admin 권한 구조 전체 개요
- **4. 시스템 연동 Flow**: Client → FE → BE → DB Flow
- **Appendix B: Common 패키지 구조**: 공유 코드 상세

Common 패키지 활용 방법:

- **검증 함수**: `isValidEmail`, `isValidPassword`, `isValidName`
- **ErrorCode 체계**: 11xxx-22xxx 범위
- **API URL 상수**: `FULL_API_URLS`
- **관리자 역할 코드**: `CODE_SYS_ADMIN_ROLES`

전체 시스템 연동 Flow (프로젝트 아키텍처 가이드 참조):

```
Client (Browser)
    |
    ▼
Frontend (React SPA)
    ├ 토큰 검증 (JWT)
    ├ 권한 체크 (UI 제어)
    └ API 호출
    |
    ▼
Backend (Express API)
    ├ 인증 체크 (authMiddleware)
    ├ 권한 체크 (adminAuthMiddleware, 역할 검증)
    └ 비즈니스 로직
    |
    ▼
Database (PostgreSQL)
    └ 데이터 CRUD
```

14.4 Backend API 규격서 참조

각 페이지에서 사용하는 API 엔드포인트:

페이지	사용 API	상세 규격서 참조
로그인	`POST /api/user/login`	API 규격서 - 인증 섹션
회원가입	`POST /api/user/register`	API 규격서 - 인증 섹션
FAQ 관리	`GET/POST/PUT/DELETE /api/admin/faq`	API 규격서 - FAQ 섹션
QnA 관리	`GET/POST/PUT/DELETE /api/admin/qna`	API 규격서 - QnA 섹션
공지사항 관리	`GET/POST/PUT/DELETE /api/admin/notice`	API 규격서 - 공지사항 섹션
사용자 관리	`GET/POST/PUT/DELETE /api/admin/user-accounts`	API 규격서 - 사용자 관리 섹션
운영자 관리	`GET/POST/PUT/DELETE /api/admin/account`	API 규격서 - 운영자 관리 섹션

페이지	사용 API	상세 규격서 참조
OpenAPI 키 관리	`GET/POST/PUT/DELETE /api/admin/open-api`	API 규격서 - OpenAPI 섹션
코드 관리	`GET/POST/PUT/DELETE /api/admin/common-code`	API 규격서 - 코드 관리 섹션

API 요청/응답 형식:

- 모든 API는 `ApiResponse<T>` 형식 사용
- ErrorCode 기반 에러 처리
- 상세 내용은 **API 규격서** 참조

본 문서는 **IITP DABT Admin Frontend 시스템**의 상세 설계를 다루고 있습니다.

추가 정보:

- **프로젝트 아키텍처 가이드**: 전체 시스템 구조 및 Common 패키지
- **Backend 상세 설계서**: Backend API 구현 및 데이터베이스 설계
- **API 규격서**: 모든 API 엔드포인트 상세 명세
- **배포 및 서버 설치 가이드**: 빌드 및 배포 상세 절차