IITP DABT Platform

설계서(자립 허브)

문서 버전: 1.0.0
작성일: 2025-11-10

(주)스위트케이

문서 History

버전	날짜	변경 내용	작성자
v1.0.0	2025-11-10	최초작성	(주)스위트케이

목차

---

1. 기본 사항

1.1 프로젝트 개요

IITP 장애인 데이터 플랫폼은 장애인 자립 지원을 위한 빅데이터 플랫폼으로, 다음과 같은 핵심 기능을 제공합니다:

- **데이터 탐색**: 자립 테마별, 데이터 유형별 데이터 목록 조회 및 검색
- **자가진단**: 장애인 자립 수준 자가진단 (4개 영역 35문항)
- **정책 추천**: 진단 결과 기반 맞춤형 정책 추천
- **정보 제공**: 자립 지원 정책, 기관, 시설 정보 제공

1.2 프로젝트 구조

```
06-IITP-DABT-Platform/
├── packages/
│   └── common/          # FE/BE 공통 타입 및 상수 관리
├── be/                  # Backend (Node.js + Express)
├── fe/                  # Frontend (React + TypeScript + Vite)
└── script/              # 빌드 및 배포 스크립트
    ├── local/           # 로컬 개발용
    └── server/          # 서버 배포용
```

1.3 핵심 특징

- **모듈화**: Common 패키지를 통한 FE/BE 타입 및 상수 공유
- **타입 안전성**: TypeScript를 통한 컴파일 타임 타입 체크
- **API_MAPPING**: 중앙 집중식 API 타입 관리
- **암호화 지원**: 민감한 환경변수 암호화 (Jasypt)

---

2. 개요

2.1 전체 프로젝트 소개

IITP 장애인 데이터 플랫폼은 **Monorepo 구조**로 구성되어 있으며, 다음과 같은 3개 모듈로 구성됩니다:

모듈	설명	주요 기술
common	FE/BE 공통 타입, 상수, API 정의	TypeScript, qs
be	RESTful API 서버	Node.js, Express, Sequelize, PostgreSQL
fe	React SPA 애플리케이션	React, Vite, TypeScript

2.2 각 모듈 개요

2.2.1 Common 모듈

- **위치**: `packages/common`
- **목적**: FE/BE 공통 타입 및 상수 관리
- **주요 기능**: API URL 및 타입 정의, 상수 정의, 자가진단 문항 및 로직 정의

2.2.2 Backend 모듈

- **위치**: `be/`
- **목적**: RESTful API 서버
- **주요 기능**: 데이터 조회 API, 자가진단 관련 API, OpenAPI 연동

2.2.3 Frontend 모듈

- **위치**: `fe/`
- **목적**: React SPA 애플리케이션
- **주요 기능**: 데이터 탐색 및 검색 UI, 자가진단 UI, 정책/기관/시설 정보 제공

---

3. 소프트웨어 아키텍처

3.1 기능 모듈 및 스펙

3.1.1 Common 패키지

기능 모듈:

- API 타입 정의 모듈
- 상수 정의 모듈
- 자가진단 로직 모듈
- 유틸리티 함수 모듈

스펙: TypeScript 5.4.0, qs 6.11.2

3.1.2 Backend

기능 모듈:

- Express 애플리케이션 설정
- 데이터 API (목록, 검색, 상세, 미리보기)
- 자가진단 API (정책, 기관, 시설)
- OpenAPI 연동
- 암호화/복호화
- 로깅 및 에러 처리

스펙: Node.js 18+, Express 4.18.2, Sequelize 6.35.2, PostgreSQL, Winston, Jasypt

3.1.3 Frontend

기능 모듈:

- 페이지 컴포넌트
- 공통 컴포넌트 (Layout, Button, Modal 등)
- API 클라이언트 및 훅
- 라우팅 (React Router)

스펙: React 18.2.0, React Router DOM 6.27.0, Vite 5.4.0, TypeScript 5.4.0

3.2 3rd 파티 라이브러리

3.2.1 Common

- `qs` : Query string 파싱

Backend

- `express` , `cors` , `helmet` , `compression`
- `winston` , `sequelize` , `pg`
- `axios` , `zod` , `date-fns` , `dotenv` , `crypto`

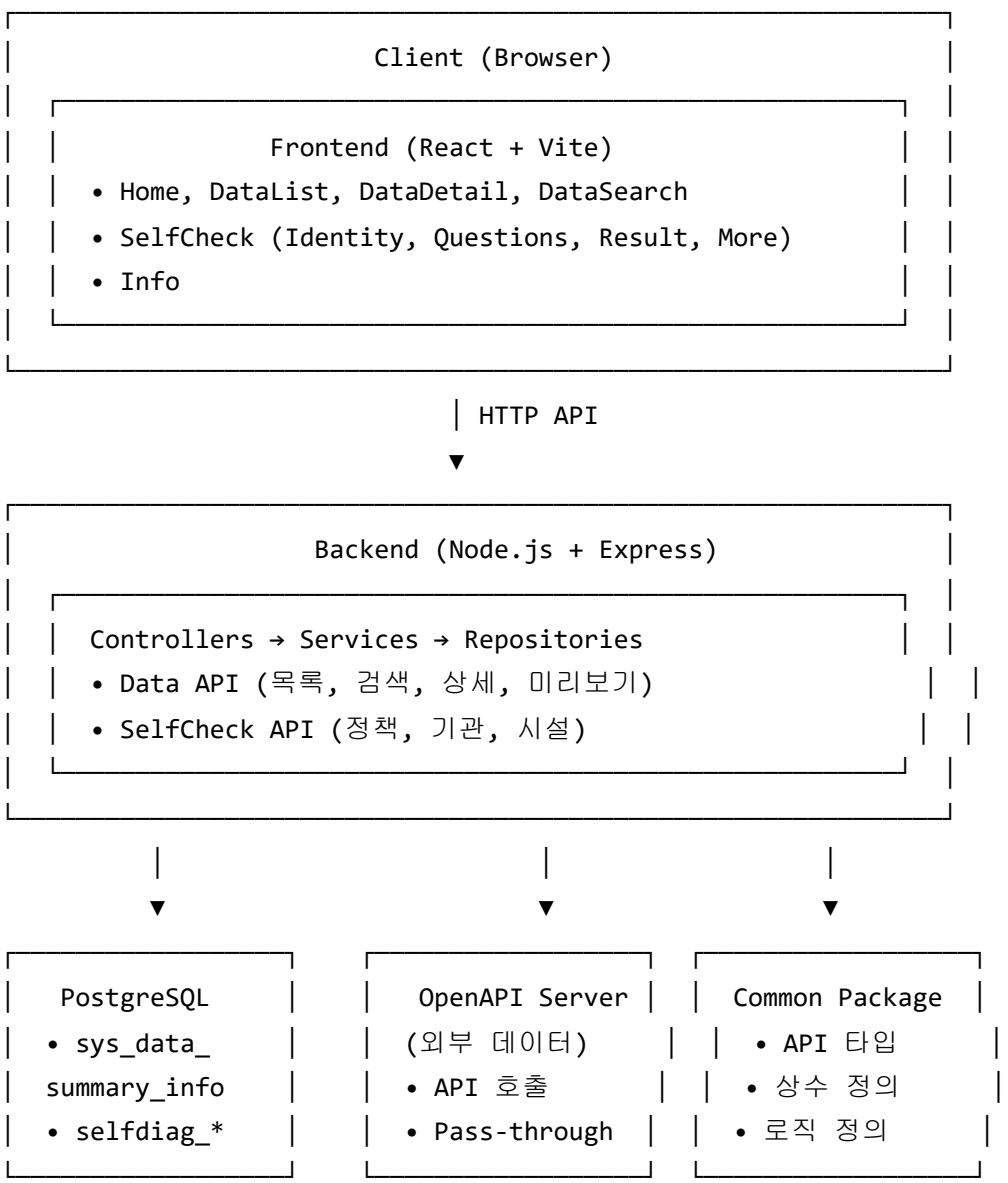
3.2.2 Frontend

- `react` , `react-dom` , `react-router-dom`

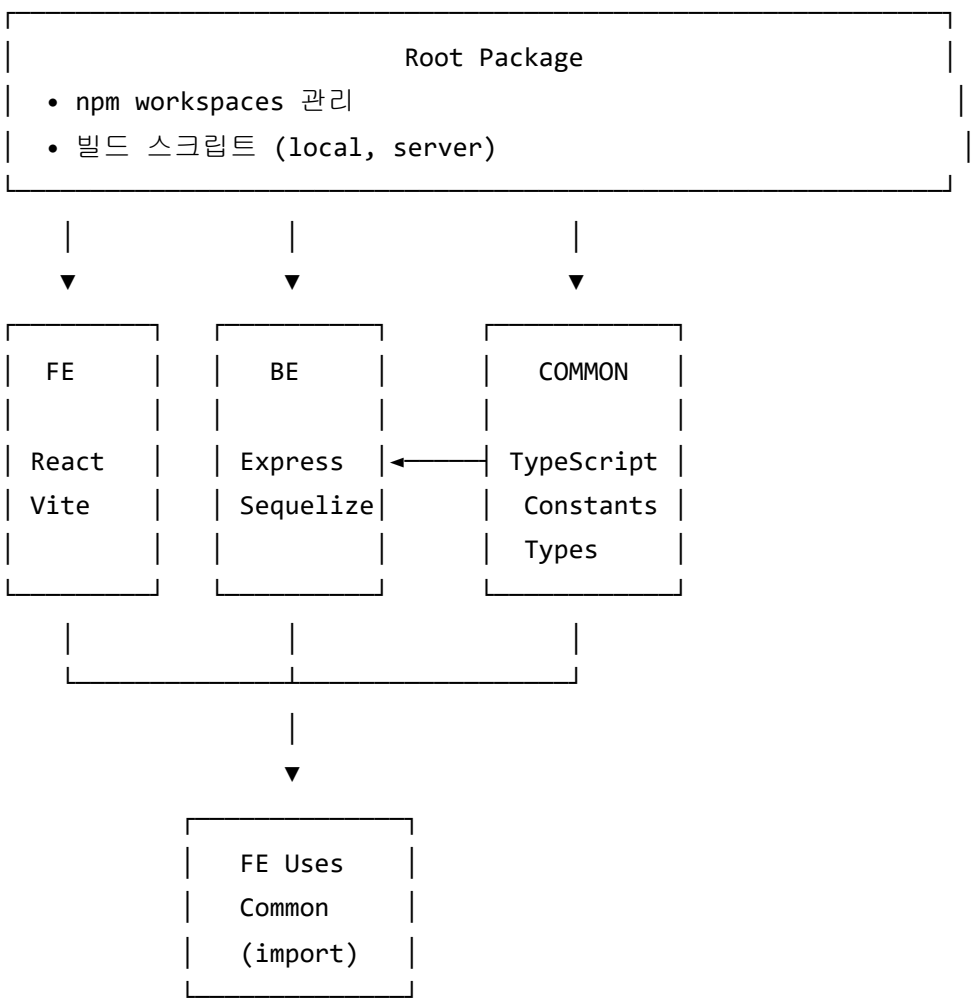
3.3 기술 스택

- **개발 언어**: TypeScript
- **개발 환경**: Node.js 18+
- **데이터베이스**: PostgreSQL
- **패키지 관리**: npm, npm workspaces
- **빌드 도구**: tsc, Vite

3.4 시스템 구성도


Client (Browser)
Frontend (React + Vite)
• Home, DataList, DataDetail, DataSearch
• SelfCheck (Identity, Questions, Result, More)
• Info
HTTP API
Backend (Node.js + Express)
Controllers → Services → Repositories
• Data API (목록, 검색, 상세, 미리보기)
• SelfCheck API (정책, 기관, 시설)
PostgreSQL
• sys_data_summary_info
• selfdiag_*
OpenAPI Server
(외부 데이터)
• API 호출
• Pass-through
Common Package
• API 타입
• 상수 정의
• 로직 정의

3.5 프로젝트 의존성 구조도

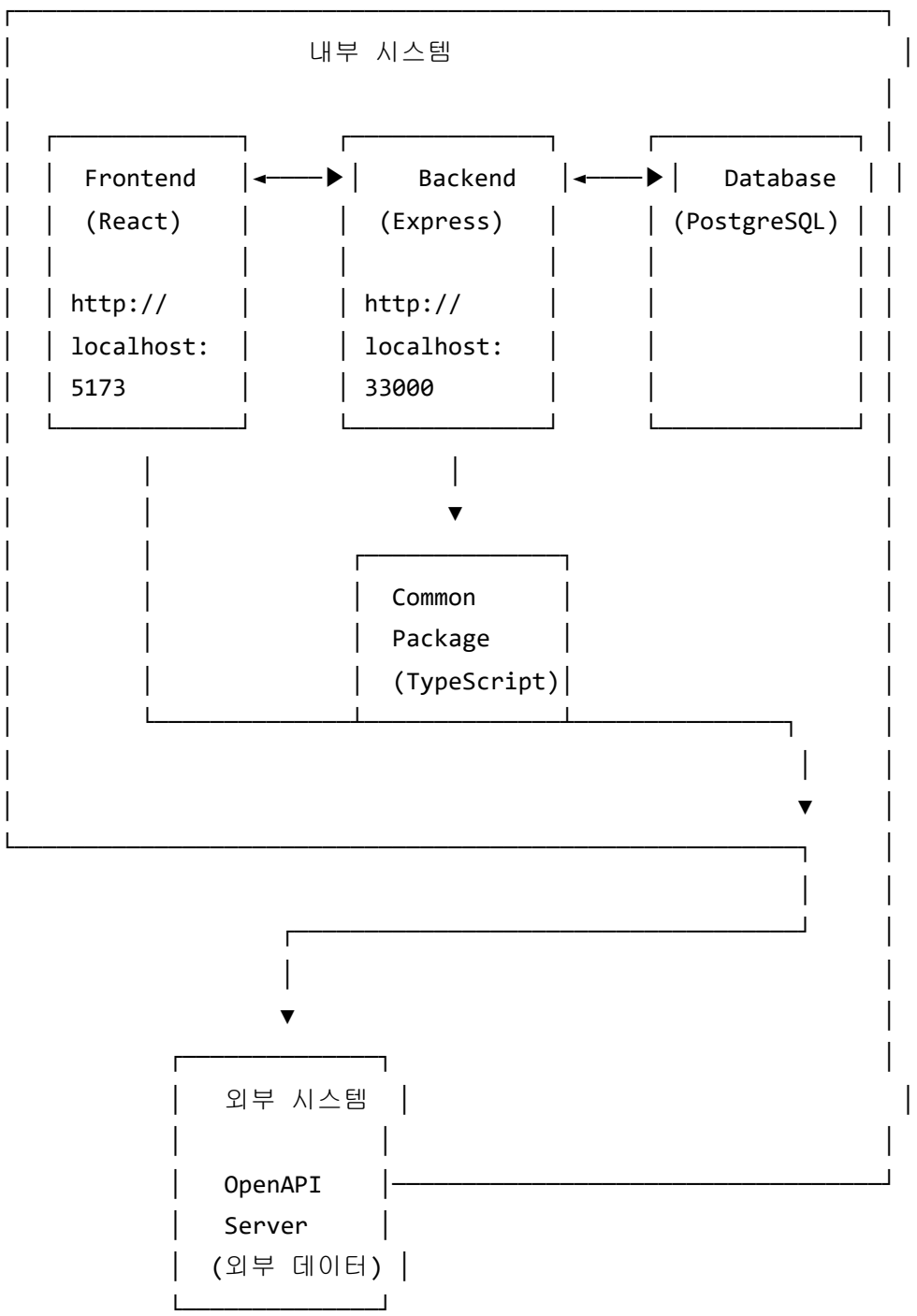


의존성 관계:

- **FE → Common**: API 타입, 상수, 유틸리티 사용
- **BE → Common**: API 타입, 상수, 유틸리티 사용
- **Common**: 독립적 모듈로 빌드 후 FE/BE에서 import

4. 전체 시스템 연동 Flow

4.1 전체 Flow 다이어그램



4.2 상세 Flow 설명

4.2.1 데이터 조회 Flow

```
Client → FE: 페이지 요청
  FE → BE: API 호출 (GET /api/v1/data/themes/phys/items)
  BE → DB: SQL 조회 (SELECT * FROM sys_data_summary_info WHERE ...)
  DB → BE: 결과 반환
  BE → FE: JSON 응답
  FE → Client: 화면 렌더링
```

4.2.2 데이터 미리보기 Flow

```
Client → FE: 미리보기 요청
  FE → BE: API 호출 (GET /api/v1/data/123/preview)
  BE → DB: 데이터 정보 조회
  DB → BE: open_api_url 반환
  BE → OpenAPI: 외부 API 호출
  OpenAPI → BE: 데이터 반환
  BE → FE: Pass-through 응답
  FE → Client: 테이블 형태로 표시
```

4.2.3 자가진단 Flow

```
Client → FE: 자가진단 시작
  FE: 로컬에서 문항 진행 (localStorage 사용)
  FE: 결과 계산 (common 패키지 로직 사용)
  FE → BE: 추천 정책 요청 (GET /api/v1/selfcheck/recommendations)
  BE → DB: 조건별 정책 조회
  DB → BE: 정책 목록 반환
  BE → FE: JSON 응답
  FE → Client: 결과 화면 표시
```

4.3 내부/외부 시스템 구분

내부 시스템:

- Frontend (React)
- Backend (Express)
- Database (PostgreSQL)

- Common Package (TypeScript)

외부 시스템:

- OpenAPI Server (외부 데이터 서버)

---

5. 전체 시스템의 기능 설명

5.1 데이터 탐색 기능

주요 기능:

1. 최신 데이터 목록 조회 (6개)
2. 테마별 데이터 건수 조회
3. 데이터 유형별 데이터 건수 조회
4. 데이터 검색 (키워드, 테마, 유형 필터)
5. 테마별 데이터 목록 조회
6. 데이터 유형별 데이터 목록 조회
7. 데이터 상세 정보 조회
8. 데이터 미리보기 조회 (OpenAPI 연동)

화면: Home, DataList, DataSearch, DataDetail

5.2 자가진단 기능

주요 기능:

1. 본인 확인 (4문항: 장애인 여부, 성별, 장애정도, 연령)
2. 자가진단 문항 (4개 영역 31문항)
3. 결과 계산 (영역별 점수, 미달 영역)
4. 정책 추천 (미달 영역 기반)
5. 정책/기관/시설 정보 제공

화면: SelfCheckStart, SelfCheckIdentity, SelfCheckQuestions, SelfCheckResult, SelfCheckMore

5.3 정보 제공 기능

주요 기능:

1. 플랫폼 소개
2. 사용 방법 안내

화면: Info

---

6. Common 모듈 상세

6.1 개요

Common 패키지는 FE와 BE가 공통으로 사용하는 타입, 상수, 로직을 중앙 집중식으로 관리하는 모듈입니다.

주요 역할:

- API 타입 정의 및 매핑
- 상수 정의 (테마, 데이터 유형, 자립 영역 등)
- 자가진단 문항 및 로직 정의
- 에러 코드 관리

6.2 소프트웨어 아키텍처

기능 모듈:

1. **API 타입 모듈**: API URL, 요청/응답 타입 정의
2. **상수 모듈**: 테마, 데이터 유형, 자립 영역 등 상수 정의
3. **자가진단 로직 모듈**: 문항, 점수 계산 로직
4. **에러 코드 모듈**: 공통 에러 코드 정의

스펙: TypeScript 5.4.0, qs 6.11.2

시스템 구성도:

```
Common Package
  ├── API Types
  │   ├── URL 정의 (API_URLS, FULL_API_URLS)
  │   ├── 요청/응답 타입
  │   └── 매핑 테이블 (API_MAPPING)
  ├── Constants
  │   ├── THEME_CONSTANTS (자립 테마)
  │   ├── DATA_TYPE_CONSTANTS (데이터 유형)
  │   └── 기타 상수
  ├── SelfCheck Logic
  │   ├── 문항 정의
  │   └── 점수 계산 로직
  └── Error Codes
      └── 에러 코드 정의
```

6.3 소스 구조 및 설명

디렉토리 구조:

```
packages/common/
├── src/
│   ├── index.ts                    # 진입점
│   └── types/
│       ├── index.ts                # 타입 export
│       ├── constants.ts            # 상수 정의
│       ├── errorCodes.ts           # 에러 코드
│       ├── selfcheck-logic.ts      # 자가진단 로직
│       └── api/
│           ├── api.ts              # API URL 정의
│           ├── common.ts           # 공통 API 타입
│           ├── data.ts             # 데이터 API 타입
│           ├── selfcheck.ts        # 자가진단 API 타입
│           └── mapping.ts          # API 매핑 테이블
├── dist/                           # 빌드 산출물
├── package.json
└── tsconfig.json
```

주요 파일:

- `src/types/constants.ts` : 테마, 데이터 유형, 자립 영역 등 상수 정의
- `src/types/selfcheck-logic.ts` : 자가진단 문항, 점수 계산 로직

- `src/types/api/mapping.ts` : API 요청/응답 타입 매핑 (body, params, query 구분)

6.4 카테고리별 기능 처리 flow

6.4.1 API 타입 관리 Flow

절차:

1. API URL 정의 (api.ts)
2. 요청/응답 타입 정의 (data.ts, selfcheck.ts)
3. API 매핑 테이블 정의 (mapping.ts)
4. FE/BE에서 import하여 사용

참조 DB Table: 없음 (타입 정의만)

6.4.2 상수 관리 Flow

절차:

1. 상수 정의 (constants.ts)
2. 유틸리티 함수 정의
3. FE/BE에서 import하여 사용

참조 DB Table: 없음 (상수만)

6.4.3 자가진단 로직 Flow

절차:

1. 문항 정의 (selfcheck-logic.ts)
2. 응답 수집 (FE)
3. 점수 계산 (FE, selfcheck-logic.ts 함수 사용)
4. 결과 판단 (FE, common 로직 사용)

참조 DB Table: 없음 (로직만)

6.5 기능 상세 설명

6.5.1 API 타입 관리

- `API_URLS` : API 경로 정의
- `FULL_API_URLS` : 완전한 URL 정의
- `API_MAPPING` : API 요청/응답 타입 매핑 (body, params, query 구분)

6.5.2 상수 관리

- 테마 코드: `phys`, `emo`, `econ`, `soc`
- 데이터 유형: `basic`, `poi`, `emp`
- 자립 영역: `basic`, `phys`, `emo`, `econ`, `soc`

6.5.3 자가진단 로직

- 문항 정의: 4개 영역 31문항
- 점수 계산: 1~5점 → 0~100점 환산
- 미달 기준: 70점 미만
- 정책 추천: 미달 영역 기반

6.6 환경 설정 및 배포

환경 설정: 없음 (타입 정의만)

빌드/배포:

- `npm run build` : TypeScript 컴파일
- `npm run dev` : Watch 모드

빌드 산출물:

- `dist/index.js` : JavaScript 파일
- `dist/index.d.ts` : 타입 정의 파일

---

7. Frontend 모듈 상세

7.1 개요

Frontend는 React + Vite로 구성된 SPA 애플리케이션으로, 사용자 인터페이스를 제공합니다.

주요 역할:

- 데이터 탐색 및 검색 UI
- 자가진단 UI (본인 확인 → 문항 → 결과 → 정책 추천)
- 정책/기관/시설 정보 제공
- API 호출 및 상태 관리

7.2 소프트웨어 아키텍처

기능 모듈:

1. **페이지 모듈**: 화면별 페이지 컴포넌트
2. **컴포넌트 모듈**: 재사용 가능한 UI 컴포넌트
3. **API 모듈**: API 클라이언트 및 훅
4. **스타일 모듈**: CSS 스타일
5. **유틸리티 모듈**: 유틸리티 함수

스펙: React 18.2.0, React Router DOM 6.27.0, Vite 5.4.0, TypeScript 5.4.0, Common Package

시스템 구성도:

```
Frontend
  ├── Pages
  │   ├── Home, DataList, DataDetail, DataSearch
  │   └── SelfCheck, Info
  ├── Components
  │   ├── Layout, Button, Modal, Table
  │   └── SelfCheck Components
  ├── API Layer
  │   ├── Client
  │   ├── Services
  │   └── Hooks
  └── Styles
      ├── globals.css
      ├── components.css
      └── data-pages.css
```

7.3 소스 구조 및 설명

디렉토리 구조:

```
fe/
├── src/
│   ├── main.tsx                        # 진입점
│   ├── pages/                          # 페이지 컴포넌트
│   │   ├── App.tsx                     # 라우터 설정
│   │   ├── Home.tsx                    # 홈 화면
│   │   ├── DataList.tsx                # 데이터 목록
│   │   ├── DataDetail.tsx              # 데이터 상세
│   │   ├── DataSearch.tsx              # 검색 결과
│   │   ├── Info.tsx                    # 정보 화면
│   │   └── self-check/                 # 자가진단 화면
│   │       ├── SelfCheckStart.tsx
│   │       ├── SelfCheckIdentity.tsx
│   │       ├── SelfCheckQuestions.tsx
│   │       ├── SelfCheckResult.tsx
│   │       └── SelfCheckMore.tsx
│   ├── components/                     # 컴포넌트
│   │   ├── layout/
│   │   │   └── Layout.tsx              # 레이아웃
│   │   ├── ui/                         # UI 컴포넌트
│   │   │   ├── Button.tsx
│   │   │   ├── Modal.tsx
│   │   │   ├── Table.tsx
│   │   │   ├── Sidebar.tsx
│   │   │   └── ... (기타)
│   │   └── self-check/                 # 자가진단 컴포넌트
│   │       ├── SelfCheckContainer.tsx
│   │       ├── SelfCheckLayout.tsx
│   │       ├── SelfCheckProgress.tsx
│   │       └── SelfCheckNavigation.tsx
│   ├── api/                            # API 레이어
│   │   ├── client.ts                   # API 클라이언트
│   │   ├── errorHandler.ts             # 에러 처리
│   │   ├── services/
│   │   │   ├── commonService.ts
│   │   │   ├── dataService.ts
│   │   │   └── selfCheckService.ts
│   │   └── hooks/
│   │       ├── useCommonApi.ts
│   │       ├── useDataApi.ts
│   │       └── useSelfCheckApi.ts
│   ├── styles/                         # 스타일
│   │   ├── globals.css                 # 전역 스타일
```

```
│   │   ├── components.css              # 컴포넌트 스타일
│   │   └── data-pages.css              # 데이터 페이지 스타일
│   └── types/
│       └── common.ts
├── public/                             # 정적 파일
├── dist/                               # 빌드 산출물
├── env.sample                          # 환경변수 샘플
├── package.json
└── vite.config.ts
```

주요 디렉토리 설명:

- `pages/` : 화면별 페이지 컴포넌트
- `components/` : 재사용 가능한 UI 컴포넌트
- `api/` : API 클라이언트, 서비스, 훅
- `styles/` : CSS 스타일 파일

7.4 페이지별 기능 설명 및 API 호출

7.4.1 Home.tsx (메인 화면)

기능:

- 최신 데이터 6개 표시
- 자립 테마별 데이터 건수 표시
- 데이터 유형별 데이터 건수 표시
- 자립 수준 자가진단 섹션
- 자립 지원 서비스 섹션

API 호출:

1. `GET /api/v1/data/summary/latest?limit=6` - 최신 데이터 조회
2. `GET /api/v1/data/counts/themes` - 테마별 데이터 건수 조회
3. `GET /api/v1/data/counts/types` - 유형별 데이터 건수 조회

주요 컴포넌트: Layout, Modal, Tag

7.4.2 DataList.tsx (데이터 목록)

기능:

- 테마별 또는 유형별 데이터 목록 표시

- 사이드바 필터 (테마, 데이터 유형)
- 페이지네이션
- 데이터 클릭 시 상세 화면 이동

API 호출:

1. `GET /api/v1/data/themes/{theme}/items` - 테마별 데이터 목록
2. `GET /api/v1/data/types/{type}/items` - 유형별 데이터 목록

주요 컴포넌트: Layout, Sidebar, FilterSection, Table

7.4.3 DataDetail.tsx (데이터 상세)

기능:

- 데이터 상세 정보 표시
- 데이터 미리보기 (OpenAPI 연동)
- 차트보기, 다운로드 버튼 (준비중)
- OpenAPI 센터 이동

API 호출:

1. `GET /api/v1/data/{id}` - 데이터 상세 정보 조회
2. `GET /api/v1/data/{id}/preview` - 데이터 미리보기 조회

주요 컴포넌트: Layout, Modal, Tag, Button

7.4.4 DataSearch.tsx (검색 결과)

기능:

- 키워드 검색 결과 표시
- 검색어 하이라이트
- 사이드바 필터

API 호출:

1. `GET /api/v1/data/search?q={keyword}` - 데이터 검색

주요 컴포넌트: Layout, Sidebar, Table

7.4.5 SelfCheckIdentity.tsx (자가진단 - 본인 확인)

기능:

- 본인 확인 문항 (4문항)
- 장애인 여부, 성별, 장애정도, 연령 선택
- localStorage에 저장

API 호출: 없음 (로컬 처리)

주요 컴포넌트: SelfCheckLayout, SelfCheckContainer

7.4.6 SelfCheckQuestions.tsx (자가진단 - 문항)

기능:

- 영역별 문항 표시 (4개 영역 31문항)
- 점수 선택 (1~5점)
- 진행 상태 표시
- localStorage에 저장

API 호출: 없음 (로컬 처리)

주요 컴포넌트: SelfCheckLayout, SelfCheckProgress, SelfCheckContainer

7.4.7 SelfCheckResult.tsx (자가진단 - 결과)

기능:

- 영역별 점수 표시
- 미달 영역 표시
- 추천 정책 표시
- 자가진단 다시하기

API 호출:

1. `GET /api/v1/selfcheck/recommendations` - 추천 정책 조회

주요 컴포넌트: SelfCheckLayout, Button

7.4.8 SelfCheckMore.tsx (자가진단 - 추가 정보)

기능:

- 자립 지원 정책 목록
- 자립 지원 기관 목록
- 자립 지원 시설 목록

API 호출:

1. `GET /api/v1/selfcheck/policies` - 정책 목록
2. `GET /api/v1/selfcheck/providers` - 기관 목록
3. `GET /api/v1/selfcheck/facilities` - 시설 목록

주요 컴포넌트: Layout, Sidebar

Info.tsx (정보 화면)

기능: 플랫폼 소개, 사용 방법 안내

API 호출: 없음

주요 컴포넌트: Layout

7.5 자가진단 로직 처리

7.5.1 자가진단 프로세스

단계:

1. **본인 확인** (SelfCheckIdentity.tsx)
   - 장애인 여부, 성별, 장애정도, 연령 선택
   - localStorage에 저장
2. **자가진단 문항** (SelfCheckQuestions.tsx)
   - 4개 영역 31문항 진행
   - 점수 선택 (1~5점)
   - 진행 상태 표시
   - localStorage에 저장
3. **결과 계산** (SelfCheckResult.tsx)
   - Common 패키지의 `calculateSelfCheckResult()` 함수 사용
   - 영역별 점수 계산
   - 미달 영역 식별 (70점 미만)
   - 정책 추천 로직 실행
4. **정책 추천** (API 호출)
   - 미달 영역 기반 정책 필터링
   - 본인 확인 정보 기반 필터링
   - 추천 정책 목록 표시

참조:

- `packages/common/src/types/selfcheck-logic.ts` : 자가진단 로직 정의
- `packages/common/src/types/constants.ts` : 상수 정의

7.5.2 자가진단 로직 흐름

```
1. 본인 확인 정보 수집
   ↓
2. 자가진단 문항 진행 (31문항)
   ↓
3. 응답 데이터 수집 (SelfCheckResponse)
   ↓
4. Common 패키지 로직으로 점수 계산
   - 영역별 점수 계산 (0~100점)
   - 미달 영역 식별 (70점 미만)
   ↓
5. 정책 추천 로직
   - 미달 영역 기반 정책 필터링
   - 개수 계산 (최대 3~4개)
   ↓
6. BE API 호출
   GET /api/v1/selfcheck/recommendations
   ↓
7. 결과 화면 표시
```

7.6 UI 컴포넌트 구성 및 CSS 구조

7.6.1 UI 컴포넌트 구성

컴포넌트 디렉토리:

- `layout/` : Layout.tsx (Header, Footer, Breadcrumb 포함)
- `ui/` : Button, Modal, Table, Sidebar, FilterSection, FilterOption, HeaderSearch, Breadcrumb, Tag, Spinner, Empty, ErrorBlock
- `self-check/` : SelfCheckLayout, SelfCheckContainer, SelfCheckProgress, SelfCheckNavigation

주요 컴포넌트:

- **Layout**: 모든 페이지 공통 레이아웃
- **Button**: variant (primary, secondary, outline), size (s, m, l) 지원
- **Modal**: 제목, 설명, 버튼(주/부) 지원, ESC/Overlay 클릭으로 닫기
- **Table**: 컬럼 정의, 로딩/에러/빈 상태 표시

7.6.2 CSS 처리 구조

CSS 파일:

- `globals.css` : 전역 스타일 (CSS Variables 정의)
- `components.css` : 컴포넌트 스타일
- `data-pages.css` : 페이지 전용 스타일

CSS Variables:

- Color System (Primary, Theme Colors)
- Typography (Pretendard 폰트, 폰트 크기/굵기/줄 높이)
- Spacing (8px 그리드 기반)
- Border Radius, Shadows, Z-Index
- Breakpoints (반응형)

처리 방식:

- CSS Variables로 디자인 시스템 일관성 유지
- Inline Styles로 동적 스타일 적용
- 1920px 최적화, 모바일 지원

7.7 Common Package 활용

TypeScript import:

```
import {
  API_URLS,
  THEME_CONSTANTS,
  DATA_TYPE_CONSTANTS,
  DataLatestItem,
  formatCount
} from '@iitp-dabt-platform/common';
```

주요 활용 영역:

1. API 호출: dataService, selfCheckService에서 Common의 타입 사용
2. 상수 사용: 테마, 데이터 유형, 자립 영역 코드
3. 유틸리티 함수: formatCount, formatDate, parseKeywords
4. 타입 안전성: Common의 타입으로 컴파일 타임 체크

7.8 환경 설정 및 배포

환경 설정 (`env.sample`):

- `VITE_API_BASE_URL` : API 서버 URL
- `VITE_API_TIMEOUT` : API 타임아웃
- `VITE_API_DATA_PREVIEW_LIMIT` : 미리보기 제한
- `VITE_VISUAL_TOOL` : 시각화 도구 URL
- `VITE_EMPLOYMENT_SITE_URL` : 구인구직 사이트 URL
- `VITE_OPEN_API_CENTER_URL` : OpenAPI 센터 URL

빌드/배포:

- `npm run build` : 프로덕션 빌드
- `npm run dev` : 개발 서버 실행
- `npm run preview` : 빌드 결과 미리보기

빌드 산출물:

- `dist/assets/` : 번들 파일
- `dist/index.html` : HTML 파일

---

8. Backend 모듈 상세

8.1 개요

Backend는 Node.js + Express로 구성된 RESTful API 서버입니다.

주요 역할:

- 데이터 조회 API
- 자가진단 관련 API
- OpenAPI 연동
- 로깅 및 에러 처리

8.2 소프트웨어 아키텍처

기능 모듈:

1. **Controller 모듈**: API 엔드포인트 처리
2. **Service 모듈**: 비즈니스 로직
3. **Repository 모듈**: 데이터 접근 계층
4. **Model 모듈**: DB Entity
5. **Middleware 모듈**: 미들웨어 (CORS, 에러 처리, 로깅 등)
6. **Utils 모듈**: 유틸리티 함수

스펙: Node.js 18+, Express 4.18.2, Sequelize 6.35.2, PostgreSQL, Winston, Jasypt

시스템 구성도:

```
Backend
  ├── Controllers
  │   ├── Data Controller
  │   └── SelfCheck Controller
  ├── Services
  │   ├── Data Service
  │   ├── SelfCheck Service
  │   └── OpenAPI Service
  ├── Repositories
  │   ├── Data Repository
  │   └── SelfCheck Repository
  ├── Models
  │   ├── DataSummaryInfo
  │   ├── SelfDiagPolicy
  │   ├── SelfDiagProvider
  │   └── SelfDiagFacility
  ├── Middleware
  │   ├── CORS
  │   ├── Error Handler
  │   ├── Logger
  │   ├── Param Converter
  │   └── Validator
  └── Utils
      ├── Date
      ├── Decrypt
      ├── Pagination
      └── Response
```

8.3 소스 구조 및 설명

디렉토리 구조:

```
be/
├── src/
│   ├── index.ts                        # 진입점
│   ├── app.ts                          # Express 앱 설정
│   ├── server.ts                       # 서버 시작
│   ├── config/                         # 설정
│   │   ├── database.ts                 # DB 연결
│   │   ├── env.ts                     # 환경변수
│   │   └── logger.ts                  # 로거
│   ├── controllers/                    # 컨트롤러
│   │   ├── common/
│   │   ├── data/
│   │   └── selfcheck/
│   ├── services/                       # 서비스
│   │   ├── data/
│   │   ├── selfcheck/
│   │   └── openapi/
│   ├── repositories/                   # Repository
│   │   ├── base/
│   │   ├── data/
│   │   └── selfcheck/
│   ├── models/                         # 모델
│   │   ├── data/
│   │   └── selfcheck/
│   ├── routes/                         # 라우트
│   │   ├── common.ts
│   │   ├── data.ts
│   │   └── selfcheck.ts
│   ├── middleware/                     # 미들웨어
│   │   ├── cors.ts
│   │   ├── errorHandler.ts
│   │   ├── logger.ts
│   │   ├── paramConverter.ts
│   │   └── validator.ts
│   └── utils/                          # 유틸리티
│       ├── date.ts
│       ├── decrypt.ts
│       ├── enumConverter.ts
│       ├── pagination.ts
│       ├── paramConverterUtils.ts
│       ├── response.ts
│       └── validation.ts
├── dist/                               # 빌드 산출물
```

```
├── logs/                                # 로그 파일
├── env.sample                           # 환경변수 샘플
├── package.json
└── tsconfig.json
```

주요 디렉토리:

- `controllers/` : API 엔드포인트 처리
- `services/` : 비즈니스 로직
- `repositories/` : DB 접근
- `models/` : DB Entity
- `middleware/` : 미들웨어 (CORS, 에러 처리, 로깅)
- `utils/` : 유틸리티 함수

8.4 카테고리별 기능 처리 flow

8.4.1 데이터 조회 Flow

절차:

1. Controller에서 요청 수신
2. Validator에서 파라미터 검증
3. ParamConverter에서 파라미터 변환
4. Service에서 비즈니스 로직 처리
5. Repository에서 DB 조회
6. 응답 반환

참조 DB Table: sys_data_summary_info, selfdiag_data_category

예시 Flow:

1. `GET /api/v1/data/themes/phys/items`
2. `paramConverter: theme(string) → phys`
3. `dataService.getThemeItems(phys, query)`
4. `dataRepository.getThemeItems(phys, options)`
5. `SELECT * FROM sys_data_summary_info WHERE self_rlty_type = 'physical'`
6. `응답 반환`

8.4.2 데이터 미리보기 Flow (OpenAPI 연동)

절차:

1. Controller에서 요청 수신
2. Repository에서 데이터 정보 조회 (open_api_url 획득)
3. OpenAPI Service에서 외부 API 호출
4. OpenAPI 특수 처리 (템플릿 치환, limit 처리, 데이터 구조 변환)
5. 응답 Pass-through

참조 DB Table: sys_data_summary_info (open_api_url)

OpenAPI 특수 처리:

1. **템플릿 치환** (`{{P_SIZE}}`):
   - URL에 `{{P_SIZE}}` 템플릿이 있으면 → `limit` 값으로 치환
   - 예: `http://api.example.com/data?size={{P_SIZE}}` → `http://api.example.com/data?size=10`
2. **응답 구조 분석**:
   - `{ items: [...] }` 구조
   - `{ content: [], page, size, total }` 구조
   - 직접 배열 `[...]` 구조
3. **Limit 처리**:
   - **items 구조**: 항상 BE에서 limit 적용 (FE limit < 20이면 20으로 증가)
   - **content/array 구조**: 템플릿 없을 때만 BE에서 limit 적용
   - **템플릿 있을 때**: OpenAPI가 이미 limit 적용했으므로 content만 추출
4. **에러 처리**:
   - 타임아웃: 504 Gateway Timeout
   - 네트워크 에러: 503 Service Unavailable
   - 서버 에러: 원본 에러 정보 전달

예시 Flow:

```
1. GET /api/v1/data/123/preview?limit=10
2. dataRepository.getDataDetail(123) → open_api_url 획득
3. openApiUrl에 {{P_SIZE}} 템플릿 있으면 → size=10으로 치환
4. OpenAPI 서버 호출
5. 응답 구조 분석 및 limit 처리
6. 데이터 변환 후 반환 (배열 형태)
```

8.4.3 자가진단 정책 추천 Flow

절차:

1. Controller에서 요청 수신
2. Service에서 필터 조건 처리

3. Repository에서 DB 조회
4. 응답 반환

참조 DB Table: selfdiag_policy

8.5 API 상세 설명

8.5.1 데이터 API

GET /api/v1/data/summary/latest - 최신 데이터 6개 조회
GET /api/v1/data/counts/themes - 자립테마별 데이터 건수 조회
GET /api/v1/data/counts/types - 데이터 유형별 데이터 건수 조회
GET /api/v1/data/search - 데이터 검색 (키워드, 테마, 유형 필터)
GET /api/v1/data/themes - 자립 테마 메타데이터 전체 조회
GET /api/v1/data/themes/items - 전체 테마 데이터 아이템 조회
GET /api/v1/data/themes/{theme}/items - 자립 테마별 리스트 조회
GET /api/v1/data/types - 데이터 유형 메타데이터 전체 조회
GET /api/v1/data/types/items - 전체 유형 데이터 아이템 조회
GET /api/v1/data/types/{type}/items - 데이터 유형별 리스트 조회
GET /api/v1/data/{id} - 데이터 테이블 상세 정보 조회
GET /api/v1/data/{id}/preview - 데이터 테이블별 미리보기 데이터 조회
(특수: OpenAPI 외부 연동, 템플릿 치환, limit 처리)

8.5.2 자가진단 API

GET /api/v1/selfcheck/recommendations - 추천 정책 리스트 조회
GET /api/v1/selfcheck/policies - 자립 지원 정책 리스트 조회
GET /api/v1/selfcheck/providers - 자립 지원 기관 리스트 조회
GET /api/v1/selfcheck/facilities - 자립 지원 시설 리스트 조회

8.5.3 공통 API

GET /api/v1/health - 헬스 체크
GET /api/v1/version - 버전 정보 조회

8.6 DB 테이블 정보

sys_data_summary_info: 데이터 요약 정보
selfdiag_data_category: 데이터 분류 정보
selfdiag_policy: 자립 지원 정책

selfdiag_provider: 자립 지원 기관
selfdiag_facility: 자립 지원 시설

8.7 환경 설정 및 배포

환경 설정 (`env.sample`):

- `NODE_ENV` : 개발 환경
- `PORT` : 서버 포트
- `DB_*` : DB 설정
- `CORS_ORIGINS` : CORS 허용 도메인
- `OPEN_API_*` : OpenAPI 설정
- `LOG_LEVEL` , `LOG_DIR` : 로깅 설정
- `ENC_SECRET` : 암호화 키

빌드/배포:

- `npm run build` : 프로덕션 빌드
- `npm run dev` : 개발 서버 실행
- `npm start` : 프로덕션 서버 실행

빌드 산출물:

- `dist/` : TypeScript 컴파일 결과
- `build-info.json` : 빌드 정보

Appendix A: Jasypt 암호화 설정

A.1 개요

민감한 환경변수를 암호화하여 보안을 강화합니다.

A.2 암호화 방식

- **알고리즘**: AES-256-CBC
- **키 해싱**: SHA-256
- **IV**: Random 16 bytes

A.3 사용 방법

암호화 스크립트:

```
cd be
ENC_SECRET=your_secret_key node script/encrypt-env.js
```

출력: `ENC(암호화된base64문자열)`

복호화 처리: Backend에서 `.env` 파일 로드 시 자동 복호화

A.4 주의사항

1. `ENC_SECRET` 은 반드시 설정해야 합니다
2. `.env` 파일은 Git에 커밋하지 않습니다
3. 암호화 키는 안전하게 보관해야 합니다