

HWP Report Generator - 백엔드 개발자 온보딩 가이드

목차

1. [프로젝트 개요](#)
2. [기술 스택](#)
3. [아키텍처](#)
4. [개발 환경 설정](#)
5. [데이터베이스 스키마](#)
6. [API 엔드포인트](#)
7. [주요 컴포넌트](#)
8. [보고서 생성 플로우 상세](#)
9. [개발 가이드라인](#)
10. [테스트](#)
11. [트러블슈팅](#)

프로젝트 개요

HWP Report Generator는 사용자가 주제를 입력하고 Claude AI와 대화하며 한글(HWP) 형식의 보고서를 자동 생성하는 대화형 웹 시스템입니다.

핵심 기능

- 사용자 인증 및 권한 관리 (JWT 기반)
- 대화형 보고서 생성** - 주제(Topic) 기반 채팅 시스템
- Claude AI를 활용한 보고서 내용 생성 (Markdown 형식)
- Markdown → HWPX 파일 변환
- 아티팩트(Artifact) 관리 - 생성된 파일 추적 및 버전 관리
- AI 사용량 추적 (메시지별)
- 관리자 대시보드

비즈니스 플로우

v2.0 (대화형 시스템):

- 사용자가 웹 UI를 통해 대화 주제(Topic) 생성
- 사용자가 메시지를 입력하여 Claude AI와 대화
- Claude API가 Markdown 형식의 보고서 내용 생성
- Markdown 파일이 아티팩트로 저장
- 사용자 요청 시 Markdown → HWPX 변환
- 변환된 HWPX 파일 다운로드 제공
- AI 사용량 데이터 자동 추적

v1.0 (기존 단일 요청 시스템) - Deprecated:

- 사용자가 웹 UI를 통해 보고서 주제 입력
- Claude API가 구조화된 보고서 내용 생성 (제목, 요약, 배경, 본문, 결론)

3. HWP Handler가 HWPX 템플릿의 XML을 수정하여 내용 삽입
 4. 생성된 HWPX 파일을 사용자에게 다운로드 제공
 5. 사용량 데이터 저장 및 통계 제공
-

기술 스택

백엔드 프레임워크

- **FastAPI** 0.104.1 - 고성능 비동기 웹 프레임워크
- **Uvicorn** 0.24.0 - ASGI 서버

데이터베이스

- **SQLite** - 경량 관계형 데이터베이스
- 파일 위치: `backend/data/hwp_reports.db`

AI/ML

- **Anthropic Claude API** (anthropic==0.71.0)
- 모델: `claude-sonnet-4-5-20250929`

인증/보안

- **python-jose[cryptography]** 3.3.0 - JWT 토큰 생성/검증
- **passlib** 1.7.4 + **bcrypt** 4.1.2 - 비밀번호 해싱

HWP 파일 처리

- **olefile** 0.47 - OLE 파일 형식 처리
- **zipfile** (표준 라이브러리) - HWPX 압축/해제

기타

- **Pydantic** 2.5.0+ - 데이터 검증 및 설정 관리
 - **python-dotenv** 1.0.0 - 환경 변수 관리
 - **aiofiles** 23.2.1 - 비동기 파일 I/O
-

아키텍처

프로젝트 구조

```

hwp-report-generator/
└── backend/
    ├── app/
    │   ├── main.py          # FastAPI 백엔드
    │   ├── routers/
    │   │   ├── auth.py       # FastAPI 애플리케이션 엔트리포인트
    │   │   ├── admin.py      # API 라우트 핸들러
    │   │   └── topics.py     # 인증 API (회원가입, 로그인, 로그아웃)
    │   └── auth.py          # 관리자 API (사용자 관리, 통계)
    └── auth.py              # ♦ 주제(Topic) API (생성, 조회, 수정, 삭제)

```

```

    └── messages.py          # ♦ 메시지 API (생성, 조회, 삭제)
    └── artifacts.py        # ♦ 아티팩트 API (조회, 다운로드, 변환)
    └── reports.py          # 보고서 API (생성, 조회, 다운로드) - Deprecated
    └── models/
        ├── user.py          # 사용자 모델
        ├── topic.py          # ♦ 주제 모델
        ├── message.py        # ♦ 메시지 모델
        ├── artifact.py       # ♦ 아티팩트 모델
        ├── ai_usage.py       # ♦ AI 사용량 모델
        ├── transformation.py # ♦ 파일 변환 모델
        ├── report.py         # 보고서 모델 - Deprecated
        └── token_usage.py    # 토큰 사용량 모델 - Deprecated
    └── database/
        ├── connection.py    # 데이터베이스 레이어
        ├── user_db.py        # DB 연결 및 초기화
        ├── topic_db.py       # 사용자 CRUD
        ├── message_db.py    # ♦ 주제 CRUD
        ├── artifact_db.py   # ♦ 메시지 CRUD
        ├── ai_usage_db.py   # ♦ 아티팩트 CRUD
        ├── transformation_db.py # ♦ AI 사용량 CRUD
        ├── report_db.py     # 파일 변환 CRUD
        └── token_usage_db.py # 보고서 CRUD - Deprecated
    └── utils/
        ├── response_helper.py # JWT 인증 및 비밀번호 해싱
        ├── artifact_manager.py # 아티팩트 파일 관리
        ├── md_handler.py      # Markdown 파일 처리
        ├── claude_client.py   # Claude API 클라이언트
        ├── hwp_handler.py     # HWPX 파일 처리
        └── auth.py            # HWPX 템플릿 파일
    └── templates/
        └── report_template.hpx
    └── artifacts/
        └── topics/
    └── output/               # 생성된 보고서 저장 - Deprecated
    └── temp/                 # 임시 파일 디렉토리
    └── data/                 # SQLite 데이터베이스
        └── hwp_reports.db
    └── tests/                # 테스트 파일
        ├── conftest.py        # pytest fixtures
        ├── test_routers_auth.py
        ├── test_utils_auth.py
        ├── test_utils_claude_client.py
        └── test_utils_hwp_handler.py
    └── requirements.txt       # Python 패키지 의존성
    └── requirements-dev.txt  # 개발/테스트 의존성
    └── runtime.txt            # Python 버전 명시
    └── pytest.ini             # pytest 설정
    └── init_db.py             # 데이터베이스 초기화 스크립트
    └── migrate_db.py          # 데이터베이스 마이그레이션 - Deprecated
    └── migrate_to_chat.py     # 채팅 시스템 마이그레이션 스크립트
    └── BACKEND_TEST.md       # 테스트 가이드 문서
    └── MIGRATION_GUIDE.md    # 마이그레이션 가이드
    └── .env                   # 환경 변수 (보안 정보)

```

```

└── frontend/          # React 프론트엔드
    └── (프론트엔드 파일들...)

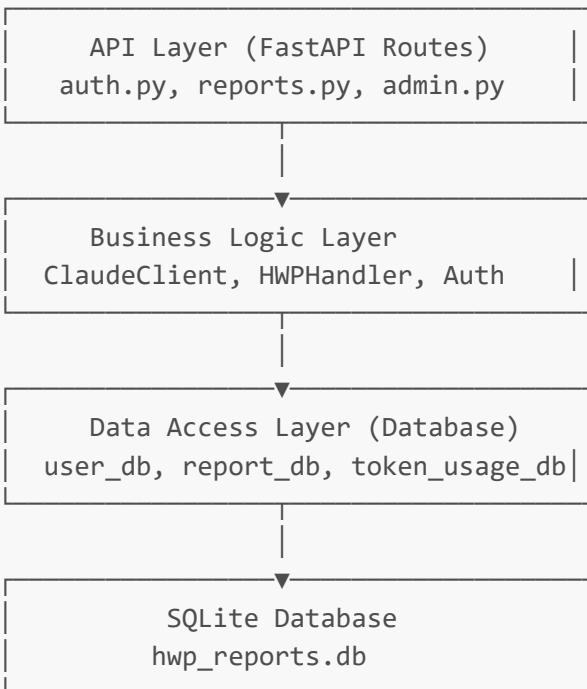
└── shared/           # ♦ 공유 모듈
    ├── models/         # 공유 데이터 모델
    │   └── api_response.py # API 응답 표준 모델
    └── constants.py    # 공유 상수 (경로 등)

└── CLAUDE.md          # 프로젝트 전체 문서
└── BACKEND_ONBOARDING.md # 이 문서

```

❖ 표시: v2.0에서 새로 추가된 파일/디렉토리 **Deprecated 표시**: v1.0 호환성 유지를 위해 남아있으나 향후 제거 예정

레이어 아키텍처



개발 환경 설정

1. 사전 요구사항

- Python 3.12+
- uv (권장) 또는 pip
- Git

2. 저장소 클론

```
git clone <repository-url>
cd hwp-report-generator
```

3. 백엔드 환경 설정

3.1 가상환경 생성 및 활성화 (uv 사용 권장)

```
cd backend
uv venv
# Windows
.venv\Scripts\activate
# macOS/Linux
source .venv/bin/activate
```

3.2 패키지 설치

```
# uv 사용 (권장)
uv pip install -r requirements.txt

# 또는 pip 사용
pip install -r requirements.txt
```

4. 환경 변수 설정

backend/.env 파일 생성:

```
# Claude API 설정
CLAUDE_API_KEY=your_api_key_here
CLAUDE_MODEL=claude-sonnet-4-5-20250929

# JWT 인증 설정
JWT_SECRET_KEY=your-secret-key-change-this-to-random-string
JWT_ALGORITHM=HS256
JWT_EXPIRE_MINUTES=1440

# 관리자 계정 설정
ADMIN_EMAIL=admin@example.com
ADMIN_PASSWORD=admin123!@#
ADMIN_USERNAME=관리자
```

보안 주의사항:

- .env 파일은 절대 Git에 커밋하지 않습니다
- JWT_SECRET_KEY는 안전한 랜덤 문자열로 변경하세요

- 프로덕션 환경에서는 강력한 비밀번호를 사용하세요

5. 데이터베이스 초기화

```
cd backend  
uv run python init_db.py
```

이 스크립트는:

- `data/` 디렉토리 생성
- SQLite 데이터베이스 생성
- 테이블 및 인덱스 생성
- 관리자 계정 생성

6. 백엔드 서버 실행

```
cd backend  
uv run uvicorn app.main:app --reload --host 0.0.0.0 --port 8000  
  
# 또는 표준 Python 사용  
uvicorn app.main:app --reload --host 0.0.0.0 --port 8000
```

7. API 문서 확인

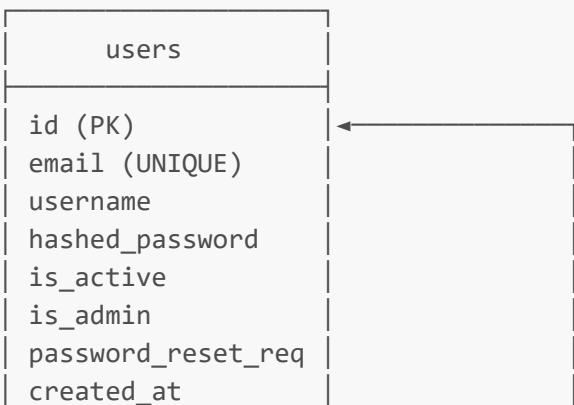
서버 실행 후 다음 URL에서 API 문서를 확인할 수 있습니다:

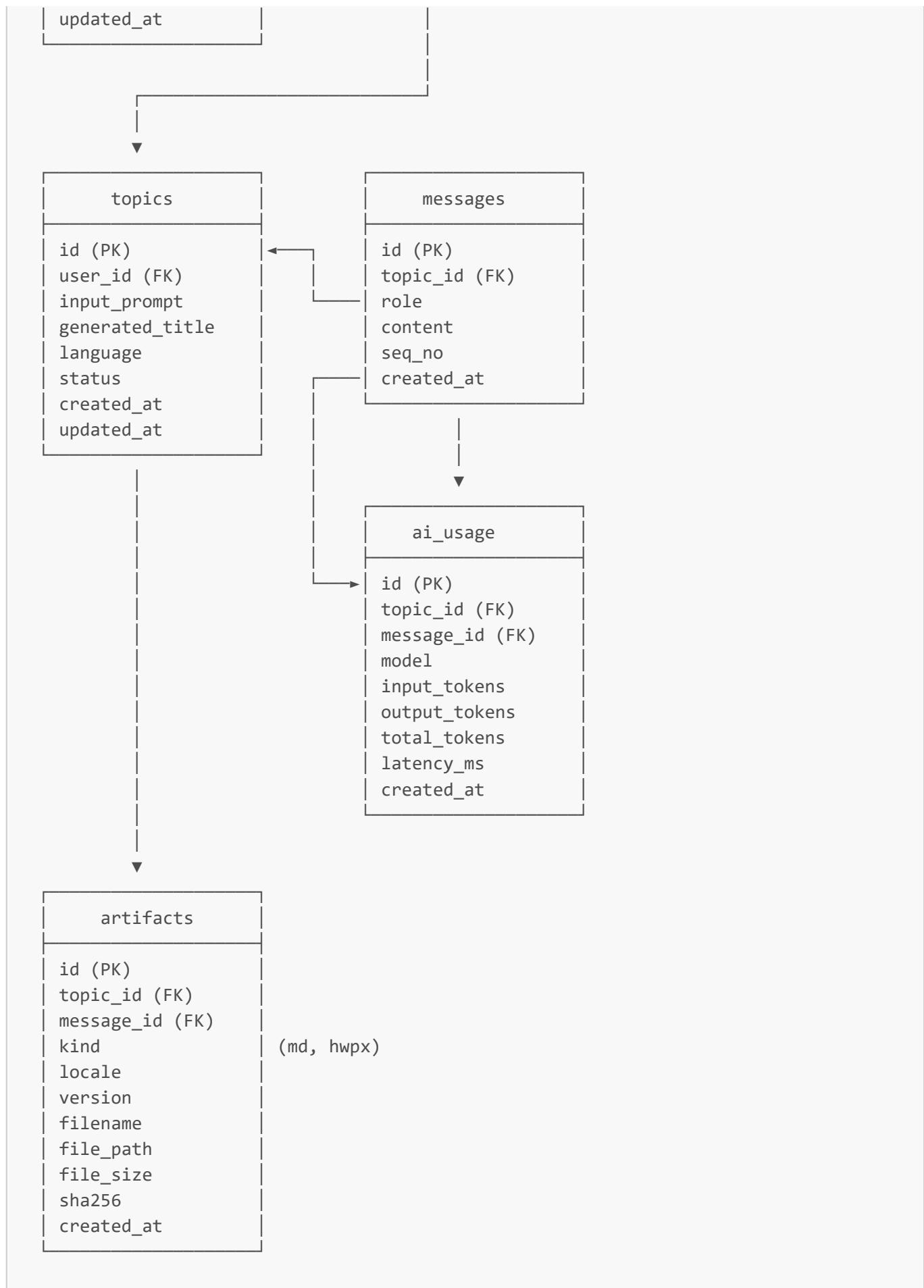
- Swagger UI: <http://localhost:8000/docs>
- ReDoc: <http://localhost:8000/redoc>

데이터베이스 스키마

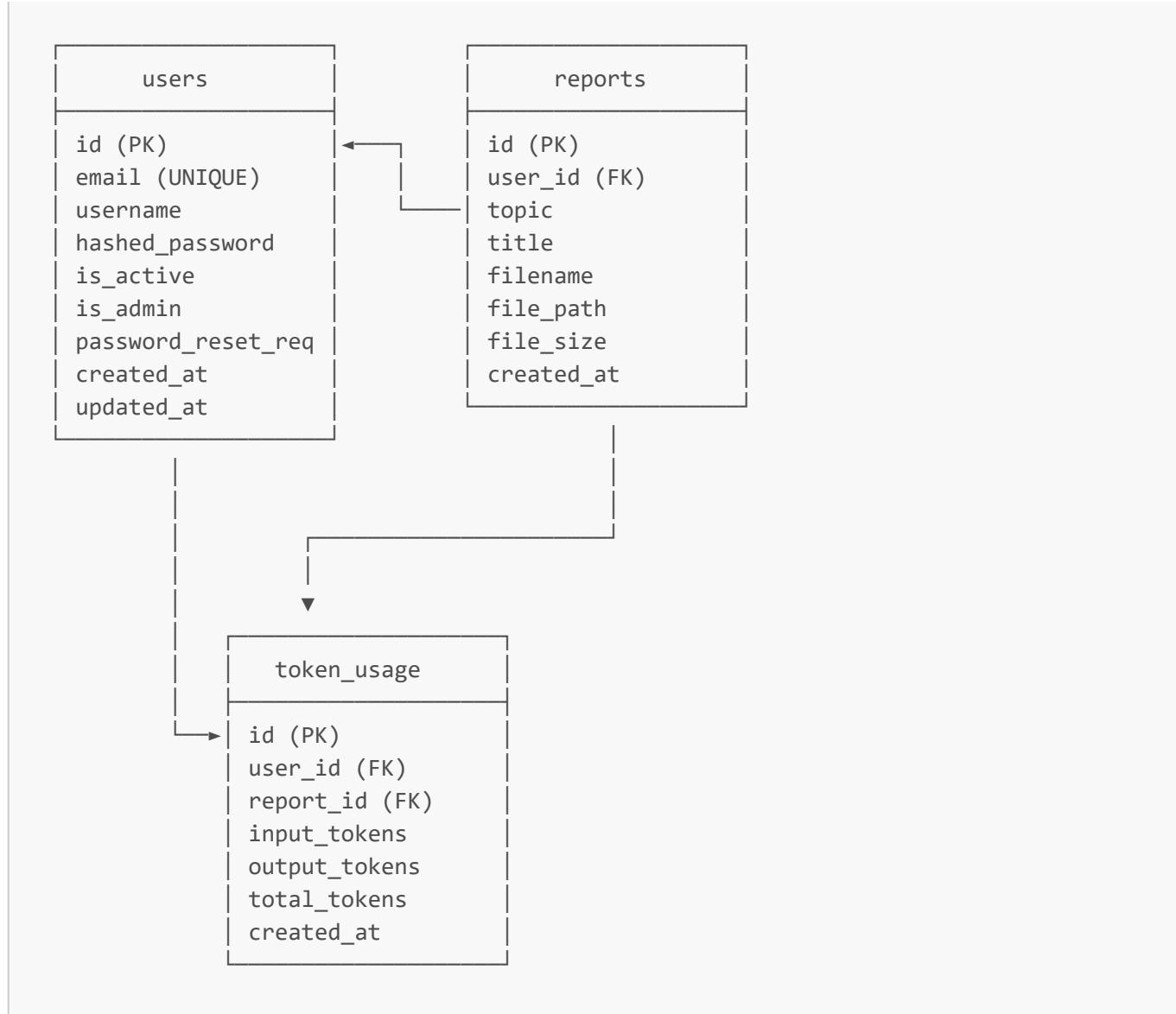
ERD (Entity Relationship Diagram)

v2.0 (대화형 시스템):





v1.0 (기존 시스템) - Deprecated:



테이블 상세

1. users 테이블

사용자 계정 정보를 저장합니다.

| 컬럼명 | 타입 | 제약조건 | 설명 |
|-----------------|---------|-------------------------------|-----------------------|
| id | INTEGER | PRIMARY KEY, AUTOINCREMENT | 사용자 고유 ID |
| email | TEXT | UNIQUE, NOT NULL | 이메일 (로그인 ID) |
| username | TEXT | NOT NULL | 사용자 이름 |
| hashed_password | TEXT | NOT NULL | 해시된 비밀번호 (bcrypt) |
| is_active | BOOLEAN | DEFAULT 0 | 계정 활성화 여부 (관리자 승인 필요) |
| is_admin | BOOLEAN | DEFAULT 0 | 관리자 권한 여부 |

| 컬럼명 | 타입 | 제약조건 | 설명 |
|-------------------------|-----------|---------------------------|----------------|
| password_reset_required | BOOLEAN | DEFAULT 0 | 비밀번호 재설정 필요 여부 |
| created_at | TIMESTAMP | DEFAULT CURRENT_TIMESTAMP | 계정 생성 시각 |
| updated_at | TIMESTAMP | DEFAULT CURRENT_TIMESTAMP | 계정 수정 시각 |

인덱스:

- idx_users_email on email

2. reports 테이블

생성된 보고서 정보를 저장합니다.

| 컬럼명 | 타입 | 제약조건 | 설명 |
|------------|-----------|----------------------------|--|
| id | INTEGER | PRIMARY KEY, AUTOINCREMENT | 보고서 고유 ID |
| user_id | INTEGER | NOT NULL, FOREIGN KEY | 작성자 사용자 ID |
| topic | TEXT | NOT NULL | 보고서 주제 (사용자 입력) |
| title | TEXT | NOT NULL | 보고서 제목 (Claude 생성) |
| filename | TEXT | NOT NULL | 파일명 (예: report_20251027_103954.hwp) |
| file_path | TEXT | NOT NULL | 파일 저장 경로 |
| file_size | INTEGER | DEFAULT 0 | 파일 크기 (바이트) |
| created_at | TIMESTAMP | DEFAULT CURRENT_TIMESTAMP | 생성 시각 |

인덱스:

- idx_reports_user_id on user_id

외래 키:

- user_id → users(id) ON DELETE CASCADE

3. token_usage 테이블

Claude API 토큰 사용량을 추적합니다.

| 컬럼명 | 타입 | 제약조건 | 설명 |
|---------|---------|----------------------------|--------------|
| id | INTEGER | PRIMARY KEY, AUTOINCREMENT | 사용량 기록 고유 ID |
| user_id | INTEGER | NOT NULL, FOREIGN KEY | 사용자 ID |

| 컬럼명 | 타입 | 제약조건 | 설명 |
|---------------|-----------|---------------------------|---------------------|
| report_id | INTEGER | FOREIGN KEY | 관련 보고서 ID (NULL 가능) |
| input_tokens | INTEGER | DEFAULT 0 | 입력 토큰 수 |
| output_tokens | INTEGER | DEFAULT 0 | 출력 토큰 수 |
| total_tokens | INTEGER | DEFAULT 0 | 총 토큰 수 |
| created_at | TIMESTAMP | DEFAULT CURRENT_TIMESTAMP | 기록 시작 |

인덱스:

- idx_token_usage_user_id on user_id

외래 키:

- user_id → users(id) ON DELETE CASCADE
- report_id → reports(id) ON DELETE SET NULL

4. topics 테이블 (v2.0) ♦◆

대화 주제/스레드 정보를 저장합니다.

| 컬럼명 | 타입 | 제약조건 | 설명 |
|-----------------|-----------|----------------------------|-----------------------------------|
| id | INTEGER | PRIMARY KEY, AUTOINCREMENT | 주제 고유 ID |
| user_id | INTEGER | NOT NULL, FOREIGN KEY | 주제 생성자 사용자 ID |
| input_prompt | TEXT | NOT NULL | 사용자가 입력한 초기 주제/프롬프트 |
| generated_title | TEXT | | AI가 생성한 대화 제목 (NULL 가능) |
| language | TEXT | NOT NULL, DEFAULT 'ko' | 언어 코드 (ko, en 등) |
| status | TEXT | NOT NULL, DEFAULT 'active' | 주제 상태 (active, archived, deleted) |
| created_at | TIMESTAMP | DEFAULT CURRENT_TIMESTAMP | 생성 시작 |
| updated_at | TIMESTAMP | DEFAULT CURRENT_TIMESTAMP | 수정 시작 |

인덱스:

- idx_topics_user_id on user_id

외래 키:

- user_id → users(id) ON DELETE CASCADE

5. messages 테이블 (v2.0) ♦◆

대화 메시지를 저장합니다 (사용자 및 AI 메시지).

| 컬럼명 | 타입 | 제약조건 | 설명 |
|-----|----|------|----|
|-----|----|------|----|

| 컬럼명 | 타입 | 제약조건 | 설명 |
|------------|-----------|----------------------------|--------------------------------|
| id | INTEGER | PRIMARY KEY, AUTOINCREMENT | 메시지 고유 ID |
| topic_id | INTEGER | NOT NULL, FOREIGN KEY | 메시지가 속한 주제 ID |
| role | TEXT | NOT NULL | 메시지 역할 ('user' 또는 'assistant') |
| content | TEXT | NOT NULL | 메시지 내용 |
| seq_no | INTEGER | NOT NULL | 주제 내 메시지 순번 (0부터 시작) |
| created_at | TIMESTAMP | DEFAULT CURRENT_TIMESTAMP | 생성 시각 |

제약 조건:

- UNIQUE (topic_id, seq_no) - 주제 내에서 seq_no는 유일해야 함

인덱스:

- idx_messages_topic_id on topic_id

외래 키:

- topic_id → topics(id) ON DELETE CASCADE

6. artifacts 테이블 (v2.0) ↗

생성된 파일(Markdown, HWPX 등)을 추적합니다.

| 컬럼명 | 타입 | 제약조건 | 설명 |
|------------|-----------|-------------------------------|---------------------------|
| id | INTEGER | PRIMARY KEY, AUTOINCREMENT | 아티팩트 고유 ID |
| topic_id | INTEGER | NOT NULL, FOREIGN KEY | 아티팩트가 속한 주제 ID |
| message_id | INTEGER | NOT NULL, FOREIGN KEY | 아티팩트를 생성한 메시지 ID |
| kind | TEXT | NOT NULL | 파일 종류 ('md', 'hwp') |
| locale | TEXT | NOT NULL, DEFAULT 'ko' | 파일 언어 |
| version | INTEGER | NOT NULL, DEFAULT 1 | 파일 버전 (같은 메시지에서 재생성 시 증가) |
| filename | TEXT | NOT NULL | 파일명 |
| file_path | TEXT | NOT NULL | 파일 저장 경로 |
| file_size | INTEGER | NOT NULL, DEFAULT 0 | 파일 크기 (바이트) |
| sha256 | TEXT | | 파일 해시 (무결성 검증용, NULL 가능) |
| created_at | TIMESTAMP | DEFAULT CURRENT_TIMESTAMP | 생성 시각 |

인덱스:

- `idx_artifacts_topic_id` on `topic_id`
- `idx_artifacts_message_id` on `message_id`

외래 키:

- `topic_id` → `topics(id)` ON DELETE CASCADE
- `message_id` → `messages(id)` ON DELETE CASCADE

7. ai_usage 테이블 (v2.0) ♦♦

메시지별 AI API 사용량을 추적합니다.

| 컬럼명 | 타입 | 제약조건 | 설명 |
|----------------------------|-----------|----------------------------|-----------------|
| <code>id</code> | INTEGER | PRIMARY KEY, AUTOINCREMENT | 사용량 기록 고유 ID |
| <code>topic_id</code> | INTEGER | NOT NULL, FOREIGN KEY | 주제 ID |
| <code>message_id</code> | INTEGER | NOT NULL, FOREIGN KEY | 메시지 ID |
| <code>model</code> | TEXT | NOT NULL | 사용된 AI 모델명 |
| <code>input_tokens</code> | INTEGER | NOT NULL, DEFAULT 0 | 입력 토큰 수 |
| <code>output_tokens</code> | INTEGER | NOT NULL, DEFAULT 0 | 출력 토큰 수 |
| <code>total_tokens</code> | INTEGER | NOT NULL, DEFAULT 0 | 총 토큰 수 |
| <code>latency_ms</code> | INTEGER | NOT NULL, DEFAULT 0 | API 응답 시간 (밀리초) |
| <code>created_at</code> | TIMESTAMP | DEFAULT CURRENT_TIMESTAMP | 기록 시각 |

인덱스:

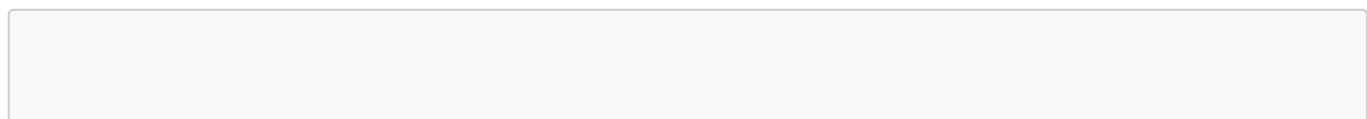
- `idx_ai_usage_topic_id` on `topic_id`
- `idx_ai_usage_message_id` on `message_id`

외래 키:

- `topic_id` → `topics(id)` ON DELETE CASCADE
- `message_id` → `messages(id)` ON DELETE CASCADE

API 엔드포인트**API Response Standard (v2.0) ♦♦**

v2.0부터 모든 API 엔드포인트는 표준화된 응답 형식을 따릅니다.

Success Response:

```
{  
  "success": true,  
  "data": { /* 실제 데이터 */ },  
  "error": null,  
  "meta": { "requestId": "uuid" },  
  "feedback": []  
}
```

Error Response:

```
{  
  "success": false,  
  "data": null,  
  "error": {  
    "code": "DOMAIN.DETAIL",  
    "httpStatus": 404,  
    "message": "에러 메시지",  
    "traceId": "uuid",  
    "hint": "해결 방법"  
  },  
  "meta": { "requestId": "uuid" },  
  "feedback": []  
}
```

자세한 내용은 프로젝트 루트의 **CLAUDE.md** 파일의 "API Response Standard" 섹션을 참조하세요.

인증 API (/api/auth)

POST /api/auth/register

사용자 회원가입

Request Body:

```
{  
  "email": "user@example.com",  
  "username": "홍길동",  
  "password": "password123"  
}
```

Response (201 Created):

```
{  
  "id": 1,  
  "email": "user@example.com",  
  "username": "홍길동",  
  "password": "password123"  
}
```

```
"username": "홍길동",
"is_active": false,
"is_admin": false,
"created_at": "2025-10-27T10:00:00",
"updated_at": "2025-10-27T10:00:00"
}
```

POST /api/auth/login

사용자 로그인

Request Body:

```
{
  "email": "user@example.com",
  "password": "password123"
}
```

Response (200 OK):

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "token_type": "bearer",
  "user": {
    "id": 1,
    "email": "user@example.com",
    "username": "홍길동",
    "is_admin": false
  }
}
```

GET /api/auth/me

현재 사용자 정보 조회 (인증 필요)

Headers:

```
Authorization: Bearer <access_token>
```

Response (200 OK):

```
{
  "id": 1,
  "email": "user@example.com",
```

```
"username": "홍길동",
"is_active": true,
"is_admin": false
}
```

POST /api/auth/logout

로그아웃 (클라이언트 측 토큰 삭제)

Response (200 OK):

```
{
  "message": "로그아웃되었습니다."
}
```

POST /api/auth/change-password

비밀번호 변경 (인증 필요)

Request Body:

```
{
  "current_password": "oldpassword123",
  "new_password": "newpassword123"
}
```

주제(Topics) API (/api/topics) (v2.0) ♦♦

POST /api/topics

새로운 대화 주제 생성 (인증 필요)

Request Body:

```
{
  "input_prompt": "2025년 디지털뱅킹 트렌드 분석",
  "language": "ko"
}
```

Response (200 OK):

```
{
  "success": true,
  "data": {
    "topic_id": "T1234567890",
    "prompt": "2025년 디지털뱅킹 트렌드 분석",
    "language": "ko"
  }
}
```

```
"id": 1,
"input_prompt": "2025년 디지털뱅킹 트렌드 분석",
"generated_title": null,
"language": "ko",
"status": "active",
"created_at": "2025-10-29T10:00:00",
"updated_at": "2025-10-29T10:00:00"
},
"error": null,
"meta": { "requestId": "..." },
"feedback": []
}
```

GET /api/topics

사용자의 주제 목록 조회 (인증 필요)

Response (200 OK):

```
{
  "success": true,
  "data": {
    "topics": [
      {
        "id": 1,
        "input_prompt": "2025년 디지털뱅킹 트렌드 분석",
        "generated_title": "디지털 뱅킹의 미래",
        "language": "ko",
        "status": "active",
        "created_at": "2025-10-29T10:00:00",
        "updated_at": "2025-10-29T10:05:00"
      }
    ],
    "total": 1
  },
  "error": null,
  "meta": { "requestId": "..." },
  "feedback": []
}
```

GET /api/topics/{topic_id}

특정 주제 조회 (인증 필요)

PATCH /api/topics/{topic_id}

주제 정보 수정 (인증 필요)

Request Body:

```
{  
  "status": "archived"  
}
```

DELETE /api/topics/{topic_id}

주제 삭제 (인증 필요)

메시지(Messages) API (/api/topics/{topic_id}/messages) (v2.0) ✨

POST /api/topics/{topic_id}/messages

주제에 새 메시지 추가 (인증 필요)

Request Body:

```
{  
  "role": "user",  
  "content": "위 주제로 상세한 보고서를 작성해주세요."  
}
```

Response (200 OK):

```
{  
  "success": true,  
  "data": {  
    "id": 1,  
    "topic_id": 1,  
    "role": "user",  
    "content": "위 주제로 상세한 보고서를 작성해주세요.",  
    "seq_no": 0,  
    "created_at": "2025-10-29T10:05:00"  
  },  
  "error": null,  
  "meta": { "requestId": "..." },  
  "feedback": []  
}
```

Note: AI 응답(role: "assistant")은 자동으로 생성되어 저장됩니다.

GET /api/topics/{topic_id}/messages

주제의 전체 메시지 조회 (인증 필요)

Response (200 OK):

```
{
  "success": true,
  "data": {
    "messages": [
      {
        "id": 1,
        "role": "user",
        "content": "...",
        "seq_no": 0,
        "created_at": "2025-10-29T10:05:00"
      },
      {
        "id": 2,
        "role": "assistant",
        "content": "...",
        "seq_no": 1,
        "created_at": "2025-10-29T10:05:10"
      }
    ],
    "total": 2
  },
  "error": null,
  "meta": { "requestId": "..." },
  "feedback": []
}
```

GET /api/topics/{topic_id}/messages/{message_id}

특정 메시지 조회 (인증 필요)

DELETE /api/topics/{topic_id}/messages/{message_id}

메시지 삭제 (인증 필요)

아티팩트(Artifacts) API (/api/artifacts) (v2.0) ⚡**GET /api/artifacts/{artifact_id}**

아티팩트 메타데이터 조회 (인증 필요)

Response (200 OK):

```
{
  "success": true,
  "data": {
    "id": 1,
    "topic_id": 1,
    "message_id": 2,
    "kind": "md",
    "content": "..."
  }
}
```

```

    "locale": "ko",
    "version": 1,
    "filename": "report_v1.md",
    "file_path": "/artifacts/topics/topic_1/messages/msg_2_v1.md",
    "file_size": 5432,
    "sha256": "abc123...",
    "created_at": "2025-10-29T10:05:10"
},
"error": null,
"meta": { "requestId": "..." },
"feedback": []
}

```

GET /api/artifacts/{artifact_id}/content

아티팩트 파일 내용 조회 (Markdown만 해당, 인증 필요)

Response (200 OK):

```

{
  "success": true,
  "data": {
    "content": "# 디지털 뱅킹의 미래\n#\n# 요약\n..."
  },
  "error": null,
  "meta": { "requestId": "..." },
  "feedback": []
}

```

GET /api/artifacts/{artifact_id}/download

아티팩트 파일 다운로드 (인증 필요)

Response:

- Content-Type: application/octet-stream
- Content-Disposition: attachment; filename="report_v1.md"
- 파일 바이너리 데이터

GET /api/artifacts/topics/{topic_id}

특정 주제의 모든 아티팩트 조회 (인증 필요)

POST /api/artifacts/{artifact_id}/convert

Markdown 아티팩트를 HWPX로 변환 (인증 필요)

Response (200 OK):

```
{  
  "success": true,  
  "data": {  
    "hpx_artifact_id": 2,  
    "hpx_filename": "report_v1.hpx",  
    "hpx_file_path": "/artifacts/topics/topic_1/messages/msg_2_v1.hpx",  
    "hpx_file_size": 45678  
  },  
  "error": null,  
  "meta": { "requestId": "..." },  
  "feedback": []  
}
```

보고서 API ([/api/reports](#)) - **Deprecated in v2.0**

⚠️ v1.0 호환성을 위해 유지되나 향후 제거 예정입니다. 새로운 코드에서는 **Topics/Messages/Artifacts API**를 사용하세요.

POST [/api/reports/generate](#)

보고서 생성 (인증 필요)

Request Body:

```
{  
  "topic": "2025년 디지털뱅킹 트렌드"  
}
```

Response (200 OK):

```
{  
  "id": 123,  
  "user_id": 1,  
  "topic": "2025년 디지털뱅킹 트렌드",  
  "title": "디지털뱅킹 혁신과 미래 전망",  
  "filename": "report_20251027_103954.hpx",  
  "file_path": "/output/report_20251027_103954.hpx",  
  "file_size": 45678,  
  "created_at": "2025-10-27T10:39:54"  
}
```

GET [/api/reports/my-reports](#)

내 보고서 목록 조회 (인증 필요)

Response (200 OK):

```
{
  "reports": [
    {
      "id": 123,
      "topic": "2025년 디지털뱅킹 트렌드",
      "title": "디지털뱅킹 혁신과 미래 전망",
      "filename": "report_20251027_103954.hwpx",
      "file_size": 45678,
      "created_at": "2025-10-27T10:39:54"
    }
  ],
  "total": 1
}
```

GET /api/reports/download/{report_id}

보고서 다운로드 (인증 필요)

Response:

- Content-Type: application/octet-stream
- Content-Disposition: attachment; filename="report_20251027_103954.hwpx"
- 파일 바이너리 데이터

관리자 API (/api/admin)**GET /api/admin/users**

전체 사용자 목록 조회 (관리자 전용)

Response (200 OK):

```
{
  "users": [
    {
      "id": 1,
      "email": "user@example.com",
      "username": "홍길동",
      "is_active": true,
      "is_admin": false,
      "created_at": "2025-10-27T10:00:00"
    }
  ],
  "total": 1
}
```

POST /api/admin/users/{user_id}/approve

사용자 승인 (관리자 전용)

Response (200 OK):

```
{  
    "message": "사용자가 승인되었습니다."  
}
```

POST /api/admin/users/{user_id}/reject

사용자 거부 (관리자 전용)

Response (200 OK):

```
{  
    "message": "사용자가 거부되었습니다."  
}
```

POST /api/admin/users/{user_id}/reset-password

사용자 비밀번호 초기화 (관리자 전용)

Response (200 OK):

```
{  
    "message": "비밀번호가 초기화되었습니다.",  
    "temporary_password": "temp123456"  
}
```

GET /api/admin/token-usage

전체 토큰 사용량 조회 (관리자 전용)

Response (200 OK):

```
{  
    "usage": [  
        {  
            "id": 1,  
            "user_id": 1,  
            "username": "홍길동",  
            "report_id": 123,  
            "input_tokens": 1500,  
            "output_tokens": 3000,  
            "total_tokens": 4500,  
            "usage_time": "2025-10-29T10:00:00Z"  
        }  
    ]  
}
```

```
        "created_at": "2025-10-27T10:39:54"
    }
],
"total_records": 1
}
```

GET /api/admin/token-usage/{user_id}

특정 사용자 토큰 사용량 조회 (관리자 전용)

주요 컴포넌트

1. response_helper ([utils/response_helper.py](#)) (v2.0) ⚡

API 응답 표준화를 위한 헬퍼 함수 모음입니다.

주요 함수:

- `success_response(data, feedback)`: 성공 응답 생성
- `error_response(code, http_status, message, details, hint)`: 에러 응답 생성
- `success_response_with_feedback(...)`: 피드백 포함 성공 응답

ErrorCode 클래스:

표준 에러 코드 상수를 정의합니다.

```
ErrorCode.AUTH_INVALID_TOKEN
ErrorCode.TOPIC_NOT_FOUND
ErrorCode.MESSAGE_CREATION_FAILED
ErrorCode.ARTIFACT_DOWNLOAD_FAILED
ErrorCode.VALIDATION_REQUIRED_FIELD
ErrorCode.SERVER_DATABASE_ERROR
```

주요 특징:

- 모든 응답에 requestId 자동 생성
- 에러 응답에 traceId 자동 생성
- 표준화된 에러 코드 관리

자세한 내용: [backend/app/utils/response_helper.py:1-173](#)

2. ArtifactManager ([utils/artifact_manager.py](#)) (v2.0) ⚡

아티팩트 파일 저장 및 관리를 담당하는 클래스입니다.

주요 메서드:

- `save_artifact(topic_id, message_id, kind, content, locale)`: 아티팩트 파일 저장

- `get_artifact_path(topic_id, message_id, kind, version, locale)`: 파일 경로 생성
- `read_artifact_content(artifact)`: 아티팩트 내용 읽기
- `calculate_sha256(file_path)`: 파일 해시 계산

파일 저장 구조:

```
backend/artifacts/
└── topics/
    └── topic_{id}/
        └── messages/
            ├── msg_{id}_v1.md
            ├── msg_{id}_v1.hwpx
            └── msg_{id}_v2.md
```

주요 특징:

- 주제 및 메시지별 디렉토리 자동 생성
- 버전 관리 지원
- UTF-8 인코딩 강제
- 파일 무결성 검증 (SHA-256)

3. MdHandler (`utils/md_handler.py`) (v2.0) ♦♦

Markdown과 HWPX 간 변환을 담당하는 핸들러 클래스입니다.

주요 메서드:

- `convert_md_to_hwpx(md_content, output_filename)`: Markdown → HWPX 변환
- `_parse_md_structure(md_content)`: Markdown 구조 파싱
- `_extract_sections(lines)`: 섹션 추출

변환 로직:

1. Markdown을 파싱하여 제목, 섹션별 내용 추출
2. 추출된 내용을 HWP 템플릿의 플레이스홀더에 매핑
3. HWPHandler를 사용하여 HWPX 파일 생성

지원 구조:

- # 제목 → {{TITLE}}
- ## 요약 → {{SUMMARY}}
- ## 배경 → {{BACKGROUND}}
- ## 주요 내용 → {{MAIN_CONTENT}}
- ## 결론 → {{CONCLUSION}}

4. ClaudeClient (`utils/clade_client.py`)

Claude API와의 통신을 담당하는 클라이언트 클래스입니다.

주요 메서드:

- `generate_report_content(topic: str)`: 주제를 입력받아 구조화된 보고서 내용 생성

반환 구조:

```
{
    "title": "보고서 제목",
    "summary": "요약 내용",
    "background": "배경 및 목적",
    "main_content": "주요 내용",
    "conclusion": "결론 및 제언",
    "usage": {
        "input_tokens": 1500,
        "output_tokens": 3000,
        "total_tokens": 4500
    }
}
```

주요 특징:

- 구조화된 프롬프트로 일관된 보고서 형식 보장
- 토큰 사용량 자동 추적
- 에러 처리 및 로깅

2. HWPHandler (`utils/hwp_handler.py`)

HWPX 파일 생성 및 수정을 담당하는 핸들러 클래스입니다.

주요 메서드:

- `create_hwp_report(content: dict, output_filename: str)`: 보고서 내용을 HWPX 파일로 생성

HWPX 파일 처리 과정:

1. HWPX 파일을 ZIP으로 압축 해제
2. XML 파일에서 플레이스홀더 검색
3. 실제 내용으로 플레이스홀더 교체
4. 줄바꿈 처리 (\n\n → 새 문단, \n → <hp:lineBreak/>)
5. 불완전한 <hp:linesegarray> 요소 제거
6. 수정된 XML을 다시 ZIP으로 압축

플레이스홀더:

- `{{TITLE}}` - 보고서 제목
- `{{DATE}}` - 생성 날짜
- `{{SUMMARY}}` - 요약
- `{{BACKGROUND}}` - 배경 및 목적
- `{{MAIN_CONTENT}}` - 주요 내용
- `{{CONCLUSION}}` - 결론 및 제언

주요 특징:

- 자동 템플릿 생성 (템플릿이 없을 경우)
- 한글 인코딩 처리 (UTF-8)
- 임시 파일 자동 정리

3. 인증 시스템 (`utils/auth.py`)

JWT 기반 인증 및 비밀번호 해싱을 담당합니다.

주요 함수:

- `hash_password(password: str)`: bcrypt로 비밀번호 해싱
- `verify_password(plain_password: str, hashed_password: str)`: 비밀번호 검증
- `create_access_token(data: dict)`: JWT 액세스 토큰 생성
- `get_current_user(token: str)`: 토큰에서 현재 사용자 정보 추출
- `get_current_active_user()`: 활성화된 사용자만 허용
- `get_current_admin_user()`: 관리자만 허용

JWT 토큰 구조:

```
{
  "sub": "user@example.com",
  "user_id": 1,
  "is_admin": false,
  "exp": 1698456789
}
```

4. 데이터베이스 레이어

각 엔티티별로 CRUD 작업을 담당하는 클래스:

UserDB (`database/user_db.py`)

- `create_user()`: 사용자 생성
- `get_user_by_email()`: 이메일로 사용자 조회
- `get_user_by_id()`: ID로 사용자 조회
- `update_user()`: 사용자 정보 수정
- `get_all_users()`: 전체 사용자 조회
- `delete_user()`: 사용자 삭제

ReportDB (`database/report_db.py`)

- `create_report()`: 보고서 생성
- `get_report_by_id()`: ID로 보고서 조회
- `get_user_reports()`: 사용자별 보고서 조회
- `get_all_reports()`: 전체 보고서 조회
- `delete_report()`: 보고서 삭제

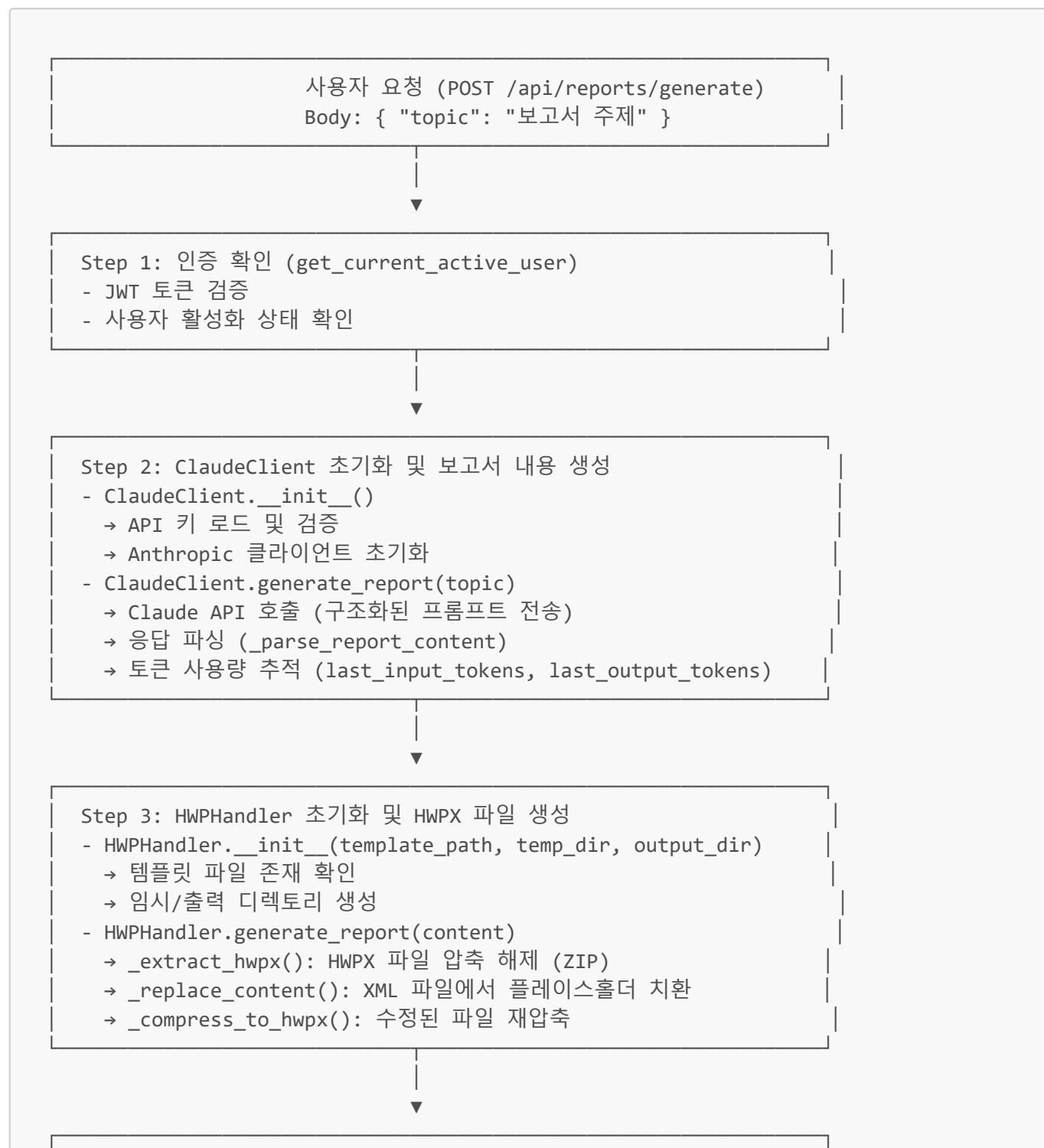
TokenUsageDB (database/token_usage_db.py)

- `create_token_usage()`: 토큰 사용량 기록
- `get_user_token_usage()`: 사용자별 토큰 사용량 조회
- `get_all_token_usage()`: 전체 토큰 사용량 조회
- `get_user_token_stats()`: 사용자별 토큰 사용 통계

보고서 생성 플로우 상세

이 섹션에서는 사용자가 보고서를 생성할 때 시스템 내부에서 일어나는 전체 프로세스를 단계별로 설명합니다.

전체 플로우 다이어그램



Step 4: 데이터베이스에 보고서 정보 저장

- ReportDB.create_report()
 - users 테이블에서 사용자 확인
 - reports 테이블에 새 레코드 삽입
 - 파일 경로, 크기, 생성 시각 등 메타데이터 저장

Step 5: 토큰 사용량 기록

- TokenUsageDB.create_token_usage()
 - token_usage 테이블에 사용량 기록
 - user_id, report_id, input_tokens, output_tokens 저장

Step 6: 응답 반환

- ReportResponse 생성
- 200 OK 응답 (보고서 ID, 파일명, 경로 등 포함)

주요 함수 상세

1. generate_report (routers/reports.py:25)

보고서 생성 API의 엔트리포인트입니다.

함수 시그니처:

```
@router.post("/generate", response_model=ReportResponse)
async def generate_report(
    request: ReportCreate,
    current_user = Depends(get_current_active_user)
) -> ReportResponse
```

파라미터:

- request: ReportCreate 모델 (topic 필드 포함)
- current_user: JWT 인증을 통해 추출된 현재 사용자 객체

처리 과정:

```
# 1. Claude 클라이언트 초기화
claude_client = ClaudeClient()

# 2. 보고서 내용 생성
content = claude_client.generate_report(request.topic)
```

```

# 3. HWP 핸들러 초기화 및 파일 생성
hwp_handler = HWPHandler(
    template_path=TEMPLATE_PATH,
    temp_dir=str(BACKEND_DIR / "temp"),
    output_dir=str(BACKEND_DIR / "output")
)
output_path = hwp_handler.generate_report(content)

# 4. 데이터베이스에 보고서 정보 저장
report = ReportDB.create_report(
    user_id=current_user.id,
    topic=request.topic,
    title=content.get("title", request.topic),
    filename=filename,
    file_path=output_path,
    file_size=file_size
)

# 5. 토큰 사용량 기록
token_usage = TokenUsageCreate(
    user_id=current_user.id,
    report_id=report.id,
    input_tokens=input_tokens,
    output_tokens=output_tokens,
    total_tokens=total_tokens
)
TokenUsageDB.create_token_usage(token_usage)

```

에러 처리:

- 모든 예외를 캐치하여 **HTTPException(500)** 반환
- 에러 메시지는 사용자에게 노출됨

코드 위치: [backend/app/routers/reports.py:25-98](#)

2. **ClaudeClient.generate_report (utils/clause_client.py:35)**

Claude API를 호출하여 보고서 내용을 생성합니다.

함수 시그니처:

```
def generate_report(self, topic: str) -> Dict[str, str]
```

파라미터:

- topic:** 사용자가 입력한 보고서 주제

반환값:

```
{
    "title": "보고서 제목",
    "title_background": "배경 섹션 제목",
    "title_main_content": "주요내용 섹션 제목",
    "title_conclusion": "결론 섹션 제목",
    "title_summary": "요약 섹션 제목",
    "summary": "요약 내용",
    "background": "배경 및 목적",
    "main_content": "주요 내용",
    "conclusion": "결론 및 제언"
}
```

처리 과정:

1. 프롬프트 생성: 구조화된 금융 보고서 작성 프롬프트 생성

- 역할 정의: "금융 기관의 전문 보고서 작성자"
- 형식 지정: 9개 섹션 (제목, 배경제목, 배경, 주요내용제목, 주요내용, 결론제목, 결론, 요약제목, 요약)
- 구분자 사용: [제목], [배경] 등의 명확한 구분자

2. Claude API 호출:

```
message = self.client.messages.create(
    model=self.model, # claude-sonnet-4-5-20250929
    max_tokens=4096,
    messages=[{"role": "user", "content": prompt}]
)
```

3. 응답 파싱: `_parse_report_content()` 호출

- 구분자를 기준으로 각 섹션 분리
- 빈 섹션이 있으면 기본 메시지로 대체

4. 토큰 사용량 저장:

```
self.last_input_tokens = message.usage.input_tokens
self.last_output_tokens = message.usage.output_tokens
self.last_total_tokens = self.last_input_tokens + self.last_output_tokens
```

로깅:

- API 호출 시작/완료 로깅
- 응답 내용 전체 로깅 (디버깅용)
- 토큰 사용량 로깅

에러 처리:

- API 호출 실패 시 예외 발생 및 로깅

코드 위치: [backend/app/utils/clause_client.py:35-125](#)

3. `ClaudeClient._parse_report_content` (`utils/clause_client.py:127`)

Claude API 응답을 파싱하여 각 섹션으로 분리합니다.

함수 시그니처:

```
def _parse_report_content(self, content: str) -> Dict[str, str]
```

파라미터:

- `content`: Claude API의 원본 응답 텍스트

처리 과정:

1. 구분자 `[제목]`, `[배경]` 등을 기준으로 문자열 분할
2. 각 섹션을 딕셔너리의 해당 키에 맵핑
3. 빈 섹션은 "(내용이 생성되지 않았습니다: {key})" 메시지로 대체

예시:

```
# 입력
"""
[제목]
디지털 뱅킹 혁신 보고서
[배경제목]
추진 배경
[배경]
최근 디지털 금융 환경의 급격한 변화로...
...
"""

# 출력
{
    "title": "디지털 뱅킹 혁신 보고서",
    "title_background": "추진 배경",
    "background": "최근 디지털 금융 환경의 급격한 변화로...",
    ...
}
```

코드 위치: [backend/app/utils/clause_client.py:127-203](#)

4. `HWPHandler.generate_report` (`utils/hwp_handler.py:34`)

HWPX 템플릿을 기반으로 보고서 파일을 생성합니다.

함수 시그니처:

```
def generate_report(  
    self,  
    content: Dict[str, str],  
    output_filename: str = None  
) -> str
```

파라미터:

- **content**: Claude가 생성한 보고서 내용 딕셔너리
- **output_filename**: 출력 파일명 (선택, 없으면 자동 생성)

반환값:

- 생성된 HWPX 파일의 절대 경로 (예: `/backend/output/report_20251027_103954.hpx`)

처리 과정:

1. 파일명 생성 (선택적):

```
if not output_filename:  
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")  
    output_filename = f"report_{timestamp}.hpx"
```

2. 임시 작업 디렉토리 생성:

```
work_dir = os.path.join(self.temp_dir, f"work_{timestamp}")  
os.makedirs(work_dir, exist_ok=True)
```

3. HWPX 압축 해제: `_extract_hpx()`

- HWPX는 ZIP 형식
- `zipfile.ZipFile`로 압축 해제

4. 내용 치환: `_replace_content()`

- Contents 디렉토리 내 모든 XML 파일 순회
- 플레이스홀더를 실제 내용으로 치환

5. 재압축: `_compress_to_hpx()`

- HWPX 표준에 따라 mimetype 파일을 먼저 압축하지 않고 추가
- 나머지 파일들은 ZIP_DEFLATED로 압축

6. 임시 디렉토리 정리:

```
finally:  
    if os.path.exists(work_dir):  
        shutil.rmtree(work_dir)
```

코드 위치: backend/app/utils/hwp_handler.py:34-76

5. HWPHandler._replace_content (utils/hwp_handler.py:89)

XML 파일에서 플레이스홀더를 실제 내용으로 치환합니다.

함수 시그니처:

```
def _replace_content(self, work_dir: str, content: Dict[str, str])
```

파라미터:

- `work_dir`: 압축 해제된 HWPX의 작업 디렉토리
- `content`: 치환할 내용 딕셔너리

처리 과정:

1. 날짜 추가:

```
content["date"] = datetime.now().strftime("%Y년 %m월 %d일")
```

2. 플레이스홀더 매핑 생성:

```
placeholders = {  
    "{{TITLE}}": content.get("title", ""),  
    "{{TITLE_BACKGROUND}}": content.get("title_background", "배경 및 목적"),  
    "{{TITLE_MAIN_CONTENT}}": content.get("title_main_content", "주요 내용"),  
    "{{TITLE_CONCLUSION}}": content.get("title_conclusion", "결론 및 제언"),  
    "{{TITLE_SUMMARY}}": content.get("title_summary", "요약"),  
    "{{SUMMARY}}": content.get("summary", ""),  
    "{{BACKGROUND}}": content.get("background", ""),  
    "{{MAIN_CONTENT}}": content.get("main_content", ""),  
    "{{CONCLUSION}}": content.get("conclusion", ""),  
    "{{DATE}}": content.get("date", "")  
}
```

3. 모든 XML 파일 순회:

```

for root_dir, dirs, files in os.walk(contents_dir):
    for filename in files:
        if filename.endswith('.xml'):
            file_path = os.path.join(root_dir, filename)
            self._replace_in_file(file_path, placeholders)

```

코드 위치: backend/app/utils/hwp_handler.py:89-127

6. HWPHandler._replace_in_file (utils/hwp_handler.py:129)

개별 XML 파일에서 플레이스홀더를 치환합니다.

함수 시그니처:

```
def _replace_in_file(self, file_path: str, placeholders: Dict[str, str])
```

파라미터:

- **file_path:** XML 파일 경로
- **placeholders:** 치환할 플레이스홀더 딕셔너리

처리 과정:

1. 파일 읽기 (UTF-8):

```

with open(file_path, 'r', encoding='utf-8') as f:
    content = f.read()

```

2. 플레이스홀더 치환:

```

for placeholder, value in placeholders.items():
    if placeholder in content:
        value_formatted = self._format_for_hwp(value)
        content = content.replace(placeholder, value_formatted)
        modified = True

```

3. linesegarray 정리: _clean_linesegarray()

- 한글 워드프로세서가 자동 계산하도록 불완전한 레이아웃 정보 제거

4. 파일 쓰기 (변경 사항이 있는 경우만):

```

if modified:
    content = self._clean_linesegarray(content)
    with open(file_path, 'w', encoding='utf-8') as f:
        f.write(content)

```

에러 처리:

- XML 파싱 에러는 무시 (바이너리 파일일 수 있음)

코드 위치: backend/app/utils/hwp_handler.py:129-162

7. HWPHandler._format_for_hwp (utils/hwp_handler.py:187)

텍스트를 HWP XML 형식에 맞게 포맷팅합니다.

함수 시그니처:

```
def _format_for_hwp(self, text: str) -> str
```

파라미터:

- text:** 포맷팅할 원본 텍스트

반환값:

- HWP XML 형식으로 포맷팅된 텍스트

처리 과정:

1. 단락 분리 (이중 줄바꿈 기준):

```
paragraphs = text.split('\n\n')
```

2. 각 단락 처리:

- 단일 줄바꿈을 임시 마커로 변경: \n → SINGLE_LINEBREAK
- XML 특수 문자 이스케이프:

```

para = para.replace('&', '&amp;')
para = para.replace('<', '&lt;')
para = para.replace('>', '&gt;')
para = para.replace('"', '&quot;')
para = para.replace("'", '&apos;')

```

- 단일 줄바꿈을 <hp:lineBreak/> 태그로 변환

3. 단락 연결:

- 여러 단락을 별도의 `<hp:p>` 태그로 분리
- 패턴: `</hp:t></hp:run></hp:p><hp:p ...><hp:run ...><hp:t>`

예시:

```
# 입력
"첫 번째 문단입니다.\n이것은 같은 문단의 다음 줄입니다.\n\n두 번째 문단입니다."

# 출력
"첫 번째 문단입니다.<hp:lineBreak/>이것은 같은 문단의 다음 줄입니다.</hp:t></hp:run>
</hp:p><hp:p id=\"2147483648\" paraPrIDRef=\"8\" styleIDRef=\"0\" pageBreak=\"0\" columnBreak=\"0\" merged=\"0\"><hp:run charPrIDRef=\"21\"><hp:t>두 번째 문단입니다."
```

코드 위치: `backend/app/utils/hwp_handler.py:187-236`

8. HWPHandler._clean_linesegarray (utils/hwp_handler.py:164)

불완전한 레이아웃 정보를 제거합니다.

함수 시그니처:

```
def _clean_linesegarray(self, content: str) -> str
```

파라미터:

- `content`: XML 내용

반환값:

- `linesegarray`가 정리된 XML 내용

처리 배경:

- 한글 워드프로세서는 `<hp:linesegarray>`가 없으면 파일을 열 때 자동으로 계산
- 하지만 불완전한 `linesegarray`가 있으면 그대로 사용하여 줄바꿈이 제대로 표시되지 않음
- 따라서 우리가 생성한 단락의 `linesegarray`를 제거하여 자동 계산하도록 함

처리 과정:

```
import re

# 패턴: </hp:t></hp:run><hp:linesegarray>...</hp:linesegarray></hp:p>
# → </hp:t></hp:run></hp:p>로 변경
pattern = r'(</hp:t></hp:run>)<hp:linesegarray>.*?</hp:linesegarray>(</hp:p>)'
```

```
content = re.sub(pattern, r'\1\2', content, flags=re.DOTALL)

return content
```

코드 위치: backend/app/utils/hwp_handler.py:164-185

9. ReportDB.create_report (database/report_db.py)

데이터베이스에 보고서 정보를 저장합니다.

함수 시그니처:

```
@staticmethod
def create_report(
    user_id: int,
    topic: str,
    title: str,
    filename: str,
    file_path: str,
    file_size: int
) -> Report
```

파라미터:

- `user_id`: 작성자 사용자 ID
- `topic`: 보고서 주제 (사용자 입력)
- `title`: 보고서 제목 (Claude 생성)
- `filename`: 파일명
- `file_path`: 파일 저장 경로
- `file_size`: 파일 크기 (바이트)

반환값:

- 생성된 `Report` 객체 (`id, created_at` 포함)

처리 과정:

```
conn = get_db_connection()
cursor = conn.cursor()

cursor.execute("""
    INSERT INTO reports (user_id, topic, title, filename, file_path, file_size)
    VALUES (?, ?, ?, ?, ?, ?)
""", (user_id, topic, title, filename, file_path, file_size))

report_id = cursor.lastrowid
conn.commit()
conn.close()
```

```
return Report(id=report_id, ...)
```

외래 키:

- user_id → users(id) ON DELETE CASCADE
-

10. TokenUsageDB.create_token_usage (database/token_usage_db.py)

토ken 사용량을 데이터베이스에 기록합니다.

함수 시그니처:

```
@staticmethod  
def create_token_usage(usage: TokenUsageCreate) -> TokenUsage
```

파라미터:

- usage: TokenUsageCreate 모델
 - user_id: 사용자 ID
 - report_id: 보고서 ID (선택)
 - input_tokens: 입력 토큰 수
 - output_tokens: 출력 토큰 수
 - total_tokens: 총 토큰 수

반환값:

- 생성된 TokenUsage 객체

처리 과정:

```
conn = get_db_connection()  
cursor = conn.cursor()  
  
cursor.execute("""  
    INSERT INTO token_usage (user_id, report_id, input_tokens, output_tokens,  
    total_tokens)  
    VALUES (?, ?, ?, ?, ?)  
""", (usage.user_id, usage.report_id, usage.input_tokens, usage.output_tokens,  
      usage.total_tokens))  
  
usage_id = cursor.lastrowid  
conn.commit()  
conn.close()  
  
return TokenUsage(id=usage_id, ...)
```

용도:

- 과금 및 사용량 추적
- 관리자 대시보드 통계
- 사용자별 사용 현황 모니터링

플로우 실행 예시

요청:

```
POST /api/reports/generate
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Content-Type: application/json

{
    "topic": "2025년 디지털 뱅킹 트렌드"
}
```

Step 1: 인증 확인

```
# JWT 토큰에서 사용자 정보 추출
current_user = User(id=1, email="user@example.com", ...)
```

Step 2: Claude API 호출

```
# ClaudeClient.generate_report() 실행
content = {
    "title": "디지털 뱅킹 혁신과 미래 전망",
    "title_background": "추진 배경",
    "background": "최근 디지털 금융 환경의 급격한 변화...",
    "title_main_content": "주요 혁신 동향",
    "main_content": "1. AI 기반 개인화 서비스...",
    "title_conclusion": "향후 과제",
    "conclusion": "지속적인 기술 투자와...",
    "title_summary": "핵심 요약",
    "summary": "디지털 뱅킹은 AI와 빅데이터를..."
}

# 토큰 사용량
claude_client.last_input_tokens = 1500
claude_client.last_output_tokens = 3200
```

Step 3: HWPX 파일 생성

```
# HWPHandler.generate_report() 실행
output_path = "/backend/output/report_20251027_103954.hpx"

# 내부 처리:
# 1. 압축 해제: /backend/temp/work_20251027_103954/
# 2. XML 수정: Contents/section0.xml 등
# 3. 재압축: report_20251027_103954.hpx
# 4. 임시 디렉토리 삭제
```

Step 4: DB 저장

```
# ReportDB.create_report()
report = Report(
    id=123,
    user_id=1,
    topic="2025년 디지털 뱅킹 트렌드",
    title="디지털 뱅킹 혁신과 미래 전망",
    filename="report_20251027_103954.hpx",
    file_path="/backend/output/report_20251027_103954.hpx",
    file_size=45678,
    created_at="2025-10-27T10:39:54"
)

# TokenUsageDB.create_token_usage()
token_usage = TokenUsage(
    id=45,
    user_id=1,
    report_id=123,
    input_tokens=1500,
    output_tokens=3200,
    total_tokens=4700,
    created_at="2025-10-27T10:39:54"
)
```

응답:

```
{
  "id": 123,
  "user_id": 1,
  "topic": "2025년 디지털 뱅킹 트렌드",
  "title": "디지털 뱅킹 혁신과 미래 전망",
  "filename": "report_20251027_103954.hpx",
  "file_size": 45678,
  "created_at": "2025-10-27T10:39:54"
}
```

성능 고려사항

- Claude API 호출 시간:** 약 5-15초 (모델 및 응답 길이에 따라 변동)
- HWPX 파일 생성 시간:** 약 1-2초 (템플릿 크기에 따라 변동)
- 데이터베이스 저장:** 약 10-50ms
- 전체 처리 시간:** 약 6-20초

최적화 포인트:

- Claude API 호출은 비동기로 처리 가능 (현재는 동기)
- HWPX 파일 압축/해제 작업은 I/O bound
- 대용량 파일의 경우 스트리밍 처리 고려

에러 발생 시나리오

| 에러 시나리오 | 발생 위치 | HTTP 상태 | 처리 방법 |
|------------------|------------------------------|---------|-----------------|
| JWT 토큰 만료/무효 | 인증 미들웨어 | 401 | 재로그인 요청 |
| Claude API 키 누락 | ClaudeClient. init | 500 | .env 파일 확인 |
| Claude API 호출 실패 | ClaudeClient.generate_report | 500 | API 키 및 네트워크 확인 |
| 템플릿 파일 없음 | HWPHandler. init | 500 | 템플릿 파일 경로 확인 |
| 디스크 공간 부족 | HWPX 파일 쓰기 | 500 | 디스크 공간 확인 |
| DB 연결 실패 | ReportDB.create_report | 500 | DB 파일 권한 확인 |

개발 가이드라인

1. 코드 스타일

- PEP 8** 준수
- 함수와 변수명: `snake_case`
- 클래스명: `PascalCase`
- 상수: `UPPER_CASE`
- Docstring: Google 스타일 권장

2. 에러 핸들링

- FastAPI의 `HTTPException` 사용
- 명확한 에러 메시지 제공
- 로깅을 통한 에러 추적

예시:

```
from fastapi import HTTPException, status

if not user:
```

```

raise HTTPException(
    status_code=status.HTTP_404_NOT_FOUND,
    detail="사용자를 찾을 수 없습니다."
)

```

3. 로깅

- `logging` 모듈 사용
- 레벨: DEBUG, INFO, WARNING, ERROR, CRITICAL
- 중요한 이벤트 및 에러는 반드시 로깅

예시:

```

import logging

logger = logging.getLogger(__name__)
logger.info(f"보고서 생성 완료: {report_id}")
logger.error(f"Claude API 호출 실패: {str(e)}")

```

4. 비동기 프로그래밍

- FastAPI는 비동기 함수 지원
- I/O 바운드 작업에 `async/await` 사용
- 파일 작업: `aiofiles` 사용 권장

예시:

```

@router.get("/my-reports")
async def get_my_reports(current_user: User = Depends(get_current_active_user)):
    reports = ReportDB.get_user_reports(current_user.id)
    return {"reports": reports}

```

5. 데이터 검증

- Pydantic 모델로 요청/응답 검증
- 모든 입력 데이터는 검증 필수

예시:

```

from pydantic import BaseModel, EmailStr, Field

class UserCreate(BaseModel):
    email: EmailStr
    username: str = Field(..., min_length=2, max_length=50)
    password: str = Field(..., min_length=8)

```

6. 보안 고려사항

- 환경 변수로 민감 정보 관리 (.env)
- 비밀번호는 반드시 해싱 후 저장
- JWT 토큰 만료 시간 설정
- CORS 설정 확인
- SQL Injection 방지 (SQLite 파라미터화 쿼리 사용)

7. API 응답 표준 (계획 중)

프로젝트는 표준화된 API 응답 형식을 정의하고 있습니다. ([CLAUDE.md](#) 참조)

Success Response:

```
{  
    "success": true,  
    "data": {  
        /* 실제 데이터 */  
    },  
    "error": null,  
    "meta": { "requestId": "..." },  
    "feedback": []  
}
```

Error Response:

```
{  
    "success": false,  
    "data": null,  
    "error": {  
        "code": "DOMAIN.DETAIL",  

```

테스트

테스트 환경 설정

테스트 프레임워크

백엔드 테스트는 pytest 기반으로 구성되어 있습니다.

설치된 테스트 도구:

- **pytest** 8.3.4 - 메인 테스트 프레임워크
- **pytest-cov** 6.0.0 - 코드 커버리지 측정
- **pytest-asyncio** 0.24.0 - 비동기 테스트 지원
- **pytest-mock** 3.14.0 - 모킹 기능
- **httpx** 0.27.2 - FastAPI TestClient용 HTTP 클라이언트

테스트 의존성 설치

```
cd backend  
uv pip install -r requirements-dev.txt
```

테스트 파일 구조

```
backend/  
└── tests/  
    ├── conftest.py          # 공통 fixtures 정의  
    ├── test_utils_auth.py    # 인증 유틸리티 테스트 (8개)  
    ├── test_routers_auth.py  # 인증 API 테스트 (6개)  
    └── (향후 추가될 테스트 파일들)  
    ├── pytest.ini            # pytest 설정 파일  
    └── requirements-dev.txt  # 개발/테스트 의존성
```

테스트 실행

전체 테스트 실행

```
cd backend  
uv run pytest tests/ -v
```

특정 테스트 파일 실행

```
# 인증 유틸리티 테스트만  
uv run pytest tests/test_utils_auth.py -v  
  
# 인증 API 테스트만  
uv run pytest tests/test_routers_auth.py -v
```

특정 테스트 클래스/메서드 실행

```
# 특정 클래스  
uv run pytest tests/test_utils_auth.py::TestPasswordHashing -v  
  
# 특정 메서드  
uv run pytest tests/test_utils_auth.py::TestPasswordHashing::test_hash_password -v
```

코드 커버리지 포함 실행

```
# 터미널에 출력  
uv run pytest tests/ -v --cov=app --cov-report=term-missing  
  
# HTML 리포트 생성 (htmlcov/ 디렉토리)  
uv run pytest tests/ --cov=app --cov-report=html  
  
# XML 리포트 생성 (CI/CD용)  
uv run pytest tests/ --cov=app --cov-report=xml
```

디버그 출력 포함 실행

```
uv run pytest tests/ -v -s
```

테스트 마커

`pytest.ini`에 정의된 마커를 사용하여 특정 유형의 테스트만 선택적으로 실행할 수 있습니다.

```
# 유닛 테스트만 실행  
uv run pytest -m unit -v  
  
# API 테스트만 실행  
uv run pytest -m api -v  
  
# 인증 관련 테스트만 실행  
uv run pytest -m auth -v  
  
# 통합 테스트 제외  
uv run pytest -m "not integration" -v
```

정의된 마커:

- `unit`: 유닛 테스트
- `integration`: 통합 테스트
- `auth`: 인증 관련 테스트
- `api`: API 엔드포인트 테스트

Fixtures (conftest.py)

테스트에서 사용 가능한 주요 fixtures입니다.

test_db

- **스코프**: function (각 테스트마다)
- **용도**: 임시 SQLite 데이터베이스 생성
- **자동 정리**: 테스트 종료 후 DB 파일 자동 삭제

```
def test_example(test_db):  
    # test_db는 임시 데이터베이스 파일 경로  
    conn = get_db_connection()  
    # 테스트 수행...
```

client

- **의존성**: test_db
- **용도**: FastAPI TestClient 제공
- **사용법**: API 엔드포인트 테스트

```
def test_api_endpoint(client):  
    response = client.get("/api/some-endpoint")  
    assert response.status_code == 200
```

test_user_data / test_admin_data

테스트용 사용자 데이터 딕셔너리를 제공합니다.

```
def test_example(test_user_data):  
    email = test_user_data["email"]          # "test@example.com"  
    username = test_user_data["username"]    # "테스트사용자"  
    password = test_user_data["password"]    # "Test1234!@#"
```

create_test_user / create_test_admin

- **의존성**: test_db, test_user_data / test_admin_data
- **용도**: 테스트용 사용자/관리자 생성 및 활성화
- **반환**: User 객체

```
def test_with_user(create_test_user):  
    user = create_test_user  
    assert user.email == "test@example.com"  
    assert user.is_active == True
```

auth_headers / admin_auth_headers

- **의존성:** client, create_test_user / create_test_admin
- **용도:** JWT 인증 헤더 생성
- **반환:** {"Authorization": "Bearer <token>"} 딕셔너리

```
def test_authenticated_endpoint(client, auth_headers):  
    response = client.get("/api/auth/me", headers=auth_headers)  
    assert response.status_code == 200
```

temp_dir

임시 디렉토리를 생성하고 테스트 종료 시 자동으로 정리합니다.

```
def test_file_operations(temp_dir):  
    file_path = os.path.join(temp_dir, "test.txt")  
    # 파일 작업 수행...  
    # 테스트 종료 후 temp_dir 자동 삭제
```

현재 테스트 커버리지 (v2.0) :

전체 커버리지: 70% (목표 달성!)

테스트 현황:

- **총 테스트:** 60개 (57 passed, 3 skipped)
- **테스트 파일:** 5개

모듈별 커버리지:

- app/utils/clade_client.py: 100% (48줄 증가)
- app/routers/reports.py: 88% (50% 증가)
- app/utils/auth.py: 87%
- app/utils/hwp_handler.py: 83% (68% 증가)
- app/database/user_db.py: 69%
- app/routers/auth.py: 68%
- app/database/connection.py: 100%

v2.0 신규 모듈 (테스트 예정):

- app/utils/response_helper.py: 미측정
- app/utils/artifact_manager.py: 미측정
- app/utils/md_handler.py: 미측정
- app/routers/topics.py: 미측정
- app/routers/messages.py: 미측정

- app/routers/artifacts.py: 미측정

목표 커버리지:

- 전체: 최소 70% 이상 (달성)
- 핵심 비즈니스 로직: 90% 이상
- 유틸리티 함수: 85% 이상

주요 개선사항:

- claude_client.py: 14% → 100% (+86%)
- hwp_handler.py: 15% → 83% (+68%)
- reports.py: 38% → 88% (+50%)
- 전체: 48% → 70% (+22%)

다음 단계:

v2.0 신규 모듈에 대한 테스트 작성 필요 (Topics, Messages, Artifacts API)

테스트 작성 가이드

1. 유닛 테스트 작성

```
import pytest
from app.utils.auth import hash_password, verify_password

@pytest.mark.unit
@pytest.mark.auth
class TestPasswordHashing:
    """비밀번호 해싱 테스트"""

    def test_hash_password(self):
        """비밀번호 해싱 테스트"""
        password = "TestPassword123!"
        hashed = hash_password(password)

        assert hashed != password
        assert len(hashed) > 0
        assert hashed.startswith("$2b$")

    def test_verify_password_success(self):
        """비밀번호 검증 성공 테스트"""
        password = "TestPassword123!"
        hashed = hash_password(password)

        assert verify_password(password, hashed) is True
```

2. API 엔드포인트 테스트

```

@pytest.mark.api
@pytest.mark.auth
class TestAuthRouter:
    """인증 API 테스트"""

    def test_register_success(self, client):
        """회원가입 성공 테스트"""
        response = client.post(
            "/api/auth/register",
            json={
                "email": "newuser@example.com",
                "username": "신규사용자",
                "password": "NewUser123!@"
            }
        )

        assert response.status_code == 200
        result = response.json()
        # 응답 검증...

    def test_login_success(self, client, create_test_user, test_user_data):
        """로그인 성공 테스트"""
        response = client.post(
            "/api/auth/login",
            json={
                "email": test_user_data["email"],
                "password": test_user_data["password"]
            }
        )

        assert response.status_code == 200
        data = response.json()
        assert "access_token" in data
        assert data["token_type"] == "bearer"

```

3. 예외 처리 테스트

```

def test_decode_access_token_invalid(self):
    """잘못된 토큰 디코딩 테스트"""
    from fastapi import HTTPException

    invalid_token = "invalid.token.here"

    with pytest.raises(HTTPException) as exc_info:
        decode_access_token(invalid_token)

    assert exc_info.value.status_code == 401

```

4. 인증이 필요한 엔드포인트 테스트

```

def test_get_me(self, client, auth_headers):
    """현재 사용자 정보 조회 테스트"""
    response = client.get("/api/auth/me", headers=auth_headers)

    assert response.status_code == 200
    data = response.json()
    assert "email" in data
    assert "username" in data

def test_get_me_unauthorized(self, client):
    """인증 없이 사용자 정보 조회 실패 테스트"""
    response = client.get("/api/auth/me")

    # 인증 헤더 없을 때 403 Forbidden 반환
    assert response.status_code == 403

```

CI/CD 통합 (GitHub Actions)

.github/workflows/backend-tests.yml 파일이 자동으로 테스트를 실행합니다.

트리거 조건

- main 또는 dev 브랜치에 push
- PR 생성 시 (main 또는 dev 브랜치 대상)
- backend/** 또는 shared/** 파일 변경 시

실행 단계

1. Python 3.12 환경 설정
2. uv 패키지 매니저 설치
3. 의존성 설치 (requirements.txt, requirements-dev.txt)
4. 환경 변수 설정 (테스트용 API 키 등)
5. pytest 실행 (코드 커버리지 포함)
- 6.Codecov에 커버리지 리포트 업로드

테스트 환경 변수

```

env:
  PATH_PROJECT_HOME: ${{ github.workspace }}
  CLAUDE_API_KEY: test_api_key
  JWT_SECRET_KEY: test_secret_key
  JWT_ALGORITHM: HS256
  JWT_EXPIRE_MINUTES: 1440

```

새 기능 개발 시 테스트 작성 프로세스

1. 테스트 파일 생성

```
cd backend/tests  
touch test_new_feature.py
```

2. 테스트 케이스 작성 (TDD 권장)

- 성공 시나리오
- 실패 시나리오
- 경계 조건 (edge cases)
- 예외 처리

3. 기능 구현

4. 테스트 실행 및 검증

```
uv run pytest tests/test_new_feature.py -v
```

5. 커버리지 확인

```
uv run pytest tests/test_new_feature.py --cov=app.module_name --cov-report=term-missing
```

6. 리팩토링 및 최적화

테스트 모범 사례

DO

- 각 테스트는 독립적으로 실행 가능해야 함
- 테스트 이름은 명확하고 서술적으로 작성 (예: `test_register_duplicate_email`)
- Fixtures를 활용하여 중복 코드 제거
- 테스트마다 새로운 데이터베이스 사용 (`test_db fixture` 활용)
- 예외 발생 시 `pytest.raises()` 사용
- API 응답 형식을 철저히 검증
- 코드 커버리지 70% 이상 유지
- 테스트 함수에 DocString 작성 (간단한 설명)

DON'T

- 실제 프로덕션 데이터베이스 사용 금지
- 외부 API 호출 시 모킹 없이 직접 호출 금지
- 테스트 간 의존성 만들지 않기
- 테스트 실행 순서에 의존하지 않기

- 하드코딩된 시간/날짜 사용 피하기 (datetime.now() 등은 모킹)
- 하나의 테스트에서 너무 많은 것을 검증하지 않기

테스트 데이터

일반 사용자 (test_user_data)

- Email: test@example.com
- Username: 테스트사용자
- Password: Test1234!@#

관리자 (test_admin_data)

- Email: admin@example.com
- Username: 관리자
- Password: Admin1234!@#

일반적인 테스트 문제 해결

1. ModuleNotFoundError: No module named 'shared'

원인: Python이 shared 모듈을 찾지 못함

해결:

- `conftest.py`에서 PATH_PROJECT_HOME이 올바르게 설정되어 있는지 확인
- 프로젝트 루트의 `.env` 파일에 PATH_PROJECT_HOME 정의 확인

2. TypeError: Client.init() got an unexpected keyword argument 'app'

원인: httpx 버전 호환성 문제

해결:

```
uv pip install httpx==0.27.2
```

3. HTTPException이 발생하지 않거나 None을 반환하는 문제

원인: 함수가 None을 반환하는 대신 예외를 발생시킴

해결:

```
# ✗ 잘못된 방법
result = decode_access_token(invalid_token)
assert result is None

# ✓ 올바른 방법
with pytest.raises(HTTPException) as exc_info:
```

```
decode_access_token(invalid_token)
assert exc_info.value.status_code == 401
```

4. database is locked 에러

원인: 데이터베이스 연결이 제대로 닫히지 않음

해결:

- `test_db` fixture가 각 테스트마다 새로운 DB를 생성하는지 확인
- 테스트에서 명시적으로 연결을 열었다면 반드시 `close()` 호출

향후 테스트 계획

다음 모듈들에 대한 테스트 추가가 필요합니다:

우선순위 높음:

- `test_utils_claude_client.py` - Claude API 클라이언트 테스트 (모킹 필수)
- `test_utils_hwp_handler.py` - HWPX 파일 처리 테스트
- `test_routers_reports.py` - 보고서 API 테스트

우선순위 중간:

- `test_routers_admin.py` - 관리자 API 테스트
- `test_database_*.py` - 데이터베이스 레이어 테스트

목표:

- 전체 코드 커버리지 70% 이상 달성
- 핵심 비즈니스 로직 90% 이상 커버리지

트러블슈팅

1. 데이터베이스 연결 오류

증상: `database is locked` 에러 발생

해결방법:

- SQLite는 단일 쓰기 잠금 사용
- 연결 후 반드시 `conn.close()` 호출
- Context manager 사용 권장:

```
with get_db_connection() as conn:
    # 작업 수행
    pass
```

2. HWPX 파일 생성 실패

증상: 생성된 파일이 한글에서 열리지 않음

해결방법:

- 템플릿 파일 무결성 확인
- XML 인코딩 확인 (UTF-8)
- 플레이스홀더 철자 확인
- `temp/` 디렉토리 권한 확인

3. Claude API 호출 실패

증상: 401 Unauthorized 또는 429 Too Many Requests

해결방법:

- `.env` 파일에 CLAUDE_API_KEY 확인
- API 키 유효성 확인
- API 사용량 제한 확인
- 네트워크 연결 확인

4. JWT 토큰 검증 실패

증상: Could not validate credentials 에러

해결방법:

- JWT_SECRET_KEY 환경 변수 확인
- 토큰 만료 시간 확인
- 토큰 형식 확인 (`Bearer <token>`)
- 클라이언트와 서버의 시간 동기화 확인

5. 파일 업로드/다운로드 오류

증상: 파일이 손상되거나 다운로드 실패

해결방법:

- output/ 디렉토리 권한 확인
- 디스크 공간 확인
- 파일 경로의 특수문자 확인
- Content-Type 헤더 확인

6. 한글 인코딩 문제

증상: 한글이 깨져서 표시됨

해결방법:

- Python 파일 인코딩: UTF-8
- XML 파일 인코딩: UTF-8
- 데이터베이스 인코딩: UTF-8
- Windows 콘솔: PYTHONIOENCODING=utf-8 환경 변수 설정

7. 마이그레이션 필요

증상: 데이터베이스 스키마 변경 필요

해결방법:

- `migrate_db.py` 스크립트 작성
- 백업 후 마이그레이션 실행:

```
cp data/hwp_reports.db data/hwp_reports.db.backup  
python migrate_db.py
```

추가 참고 자료

공식 문서

- [FastAPI 공식 문서](#)
- [Anthropic Claude API 문서](#)
- [Pydantic 문서](#)
- [SQLite 문서](#)

프로젝트 문서

- `CLAUDE.md`: 프로젝트 전체 개요 및 Claude Code 가이드
- `README.md`: 프로젝트 소개 및 빠른 시작 가이드

유용한 명령어

```
# 데이터베이스 초기화  
uv run python init_db.py  
  
# 개발 서버 실행  
uv run uvicorn app.main:app --reload --host 0.0.0.0 --port 8000  
  
# 패키지 업데이트  
uv pip install -r requirements.txt --upgrade  
  
# 환경 변수 확인  
cat .env  
  
# 로그 확인  
tail -f logs/app.log # (로그 파일이 설정된 경우)
```

연락처 및 지원

문제가 발생하거나 질문이 있으면:

1. 이슈 트래커에 등록
 2. 팀 채널에 문의
 3. 코드 리뷰 요청
-

작성일: 2025-10-27 **최종 업데이트:** 2025-10-29 (v2.0 대화형 시스템 반영) **버전:** 2.0 **담당자:** Backend Development Team

주요 변경사항 (v2.0)

시스템 아키텍처 변경

- **단일 요청 → 대화형 시스템:** 기존 단일 보고서 생성 방식에서 채팅 기반 대화형 시스템으로 전환
- **파일 형식:** 직접 HWPX 생성 → Markdown 생성 후 HWPX 변환

새로운 테이블

- `topics` - 대화 주제/스레드
- `messages` - 사용자 및 AI 메시지
- `artifacts` - 생성된 파일 (MD, HWPX)
- `ai_usage` - 메시지별 AI 사용량 추적

새로운 API

- `/api/topics` - 주제 CRUD
- `/api/topics/{topic_id}/messages` - 메시지 관리
- `/api/artifacts` - 아티팩트 조회 및 변환

새로운 유틸리티

- `response_helper.py` - API 응답 표준화
- `artifact_manager.py` - 아티팩트 파일 관리
- `md_handler.py` - Markdown ↔ HWPX 변환

API Response Standard

모든 API 엔드포인트가 표준화된 응답 형식 사용 (100% compliance)

테스트 커버리지 개선

- 48% → 70% (+22%)
- `claude_client.py`: 14% → 100%
- `hwp_handler.py`: 15% → 83%

Deprecated

- `/api/reports` - v1.0 호환성 유지, 향후 제거 예정
- `reports, token_usage` 테이블 - v1.0 호환성 유지

마이그레이션

- 마이그레이션 가이드: [backend/MIGRATION_GUIDE.md](#)
- 마이그레이션 스크립트: [backend/migrate_to_chat.py](#)