API 설계 문서 작성법

최종 프로젝트 STEP 2 · REST API 및 Function Calling 명세서

AI 애플리케이션 개발 과정 · 최종 프로젝트 산출물 가이드

`GET /api/v1/incidents` `POST /incidents/{id}/recovery` `202 Accepted` `restart_service`

API 설계 문서는 팀의 계약서다

코드를 쓰기 전에, 무엇을 주고받을지 합의해 두는 문서

프런트 ↔ 백엔드

따로 개발해도 맞물린다

화면이 어떤 주소로 무엇을 보내고 무엇을 받는지 미리 정한다. 서로 기다리지 않고 동시에 개발할 수 있다.

에이전트 ↔ 도구

Function Calling의 기준

LLM이 호출할 Tool의 이름, 입력, 실패 코드를 정한다. 에이전트 설계가 이 명세 위에 올라간다.

설계 ↔ 검증

테스트의 근거

평가는 문서가 있는지가 아니라 설계·구현·검증이 이어지는지를 본다. 명세가 곧 테스트 케이스가 된다.

STEP 2에서 이 문서가 놓이는 자리

앞 단계의 요구사항을 받아, 뒤 단계의 구현과 검증에 기준을 넘겨준다

기획서
기능 요구사항
FR-01 …

→

API 설계 문서
오늘 다루는
산출물

→

DB 스키마 설계서
STEP 2의
짝 산출물

→

에이전트 설계
Tool 정의를
가져다 쓴다

→

통합 테스트
명세대로
동작하는지 확인

API 설계 문서와 DB 스키마 설계서는 STEP 2의 필수 산출물이다.
두 문서는 같은 ID 형식을 쓰고, API 필드가 어느 DB 컬럼에 저장되는지 대응을 밝힌다.

문서에 들어가는 여섯 가지

01

공통 정보와 인증

서버 주소, 인증, 오류 형식, /docs 위치, 변경 이력

02

API 목록

메서드·경로·인증·설명을 한눈에 보는 표

03

엔드포인트 상세 명세

필드 명세 표와 요청·응답·오류 예시

04

검증과 권한 규칙

입력 형식 검증과 서버의 권한·승인 검사

05

Tool 정의와 호출 흐름

입력·출력·실패 처리·실행 조건과 순서

06

실행 규칙

타임아웃, 재시도, 중복 실행 방지

REST는 자원을 주소로, 행위를 메서드로 나타낸다

RESTful API는 이 약속을 지켜서 만든 API다. 주소만 봐도 무엇을 다루는지 알 수 있다

자원 (Resource)

다루는 대상에 주소를 준다

장애, 복구 작업처럼 서비스가 다루는 대상 하나하나가 고유한 주소(URL)를 갖는다.

```
/incidents
/incidents/inc-001
```

행위 (Method)

할 일은 메서드로 말한다

같은 주소에 메서드만 바꿔 조회, 생성, 수정, 삭제를 구분한다.

```
GET · POST
PATCH · DELETE
```

표현 (Representation)

내용은 JSON에 담는다

자원의 현재 상태를 JSON 문서로 표현해 요청과 응답에 싣는다.

```
{"id": "inc-001",
 "status": "open"}
```

REST가 지키는 세 가지 원칙

원칙을 지키면 처음 보는 API도 예측할 수 있고, 서버를 여러 대로 늘리기 쉽다

원칙 1

클라이언트와 서버의 분리

화면은 API 규격만 알면 된다. 서버 내부가 바뀌어도 규격이 같으면 화면은 그대로 동작한다.

원칙 2

무상태 (Stateless)

서버는 앞선 요청을 기억하지 않는다. 그래서 요청마다 토큰과 필요한 ID를 모두 실어 보낸다.

원칙 3

일관된 인터페이스

모든 자원에 같은 주소 규칙, 같은 메서드 의미, 같은 오류 형식을 쓴다.

HTTP 요청과 응답은 세 부분으로 읽는다

- 시작줄: 요청은 메서드와 경로, 응답은 상태 코드
- 헤더: 인증 토큰, 본문 형식 같은 부가 정보
- 본문: 주고받는 데이터(JSON). GET 요청에는 보통 본문이 없다.
- API 설계 문서는 이 세 부분을 엔드포인트마다 적는 것이다.

요청과 응답의 구조

```
# 요청 ① 시작줄
POST /api/v1/incidents/inc-001/recovery HTTP/1.1
# 요청 ② 헤더
Authorization: Bearer <JWT>
Content-Type: application/json
# 요청 ③ 본문
{"action": "restart", "approval_id": "apr-001"}

# 응답 ① 시작줄
HTTP/1.1 202 Accepted
# 응답 ② 헤더
Content-Type: application/json
# 응답 ③ 본문
{"job_id": "job-001", "status": "queued"}
```

메서드마다 의미와 성질이 정해져 있다

메서드	하는 일	안전 (조회만)	멱등	성공 코드
GET	조회	예	예	200
POST	생성, 실행 요청	아니오	아니오	201 / 202
PUT	전체 교체	아니오	예	200
PATCH	일부 수정	아니오	보장되지 않음	200
DELETE	삭제	아니오	예	204

멱등은 같은 요청을 여러 번 보내도 결과가 한 번 보낸 것과 같다는 뜻이다.
POST는 멱등이 아니므로, 복구 요청이 두 번 들어와도 한 번만 실행되게 하는 장치(중복 방지 키, 409 응답)를 설계에 넣는다.

상태 코드는 첫 자리로 읽는다

누구 쪽 문제인지가 첫 자리에 담겨 있다. 무조건 200으로 응답하고 본문에 오류를 넣지 않는다

2xx

성공

요청을 정상 처리했다. 조회, 생성, 접수, 삭제에 따라 코드를 구분한다.

```
200 OK
201 Created
202 Accepted
204 No Content
```

4xx

요청 쪽 문제

보낸 쪽이 고쳐야 한다. 값, 인증, 권한, 대상, 충돌 중 무엇이 문제인지 알려준다.

```
400 · 422 잘못된 값
401 인증 · 403 권한
404 없음 · 409 충돌
```

5xx

서버 쪽 문제

서버가 고쳐야 한다. 내부 오류 내용은 응답에 노출하지 않고 로그에 남긴다.

```
500 Internal Server Error
503 Service Unavailable
```

주소 규칙: RESTful하지 않은 주소를 고쳐 쓴다

이렇게 쓰지 않는다	이렇게 쓴다	규칙
`GET /getIncidents`	`GET /incidents`	주소는 복수형 명사, 동사는 메서드가 말한다
`POST /deleteIncident?id=1`	`DELETE /incidents/1`	대상은 경로에, 행위는 메서드에 둔다
`GET /incidents/1/delete`	`DELETE /incidents/1`	GET은 조회만 한다. 데이터를 바꾸지 않는다
`/Recovery_Jobs`	`/recovery-jobs`	소문자와 하이픈으로 통일한다
`/incidents`	`/api/v1/incidents`	버전을 붙여 규격이 바뀌어도 화면이 안 깨지게
`POST /recovery?incident=1`	`POST /incidents/1/recovery`	특정 대상에 딸린 행위는 그 대상 아래에

진단·복구처럼 '실행'하는 행위는 명사로 떨어지지 않는다.
이때는 POST와 하위 경로(/incidents/{id}/diagnose)로 표현하는 것이 실무 관례다.

목록 조회는 필터·정렬·페이지를 쿼리로 받는다

- 경로는 자원을 가리키고, 조건은 ? 뒤 쿼리 파라미터로 붙인다.
- 요구사항 FR-01의 "상태·발생시간·자산명으로 필터링"이 곧 쿼리 파라미터 목록이다.
- 응답에는 목록과 함께 전체 건수, 현재 페이지를 담는다.
- 파라미터 이름과 기본값은 팀에서 정해 문서에 적는다.

GET /api/v1/incidents (형식 예시)

```
GET /api/v1/incidents?status=open&asset=svc-api-01
    &sort=-created_at&page=1&size=20
Authorization: Bearer <JWT>

HTTP/1.1 200 OK
{
  "items": [
    {"id": "inc-001", "status": "open",
     "asset": "svc-api-01"}
  ],
  "total": 37,
  "page": 1,
  "size": 20
}
```

요구사항에서 API를 뽑아낸다

요구사항 한 줄마다 "무엇을, 어떻게 하는가"를 물으면 엔드포인트가 나온다

요구사항	다루는 대상	행위	엔드포인트
FR-01 장애 목록 조회	incidents	조회	`GET /api/v1/incidents`
FR-02 AI 원인 진단	incidents의 진단	실행 요청	`POST /api/v1/incidents/{id}/diagnose`
FR-03 복구 작업 승인	incidents의 복구	실행 요청	`POST /api/v1/incidents/{id}/recovery`
FR-04 복구 이력 조회	recovery-jobs	조회	`GET /api/v1/recovery-jobs/{id}`

예시는 산출물 샘플(InfraCare AI, 교육용 가상 서비스) 기준이다. 팀 주제의 대상과 행위로 바꿔 쓴다.

문서 머리말에 공통 정보를 모은다

- 개발 서버와 배포 서버의 기본 주소를 적는다.
- FastAPI가 만드는 자동 문서(/docs)의 위치를 적고, 문서와 대조한다.
- 오류 응답은 모든 API가 같은 형식을 쓴다.
- 바뀔 때마다 버전을 올리고 변경 이력에 남긴다.

API 설계 문서 머리말 (형식 예시)

```
# API 설계 문서 — <서비스명>
문서 버전 v0.2 | 작성 상태: 검토 중

## 공통 정보
- 개발 서버: http://localhost:8000/api/v1
- 배포 서버: https://<배포 도메인>/api/v1
- 자동 문서: <서버 주소>/docs, /openapi.json
- 인증: Authorization: Bearer <JWT>

## 공통 오류 형식
{"error": {"code": "APPROVAL_REQUIRED",
           "message": "승인이 필요합니다",
           "request_id": "req-001"}}

## 변경 이력
| v0.1 | 11/18 | 초안 작성         | 백엔드 담당 |
| v0.2 | 11/20 | 409 응답 추가     | 백엔드 담당 |
```

인증 방식은 네 줄로 적는다

항목	적을 내용	예시
인증	토큰을 어디서 발급받는가	Supabase Auth 로그인 → JWT Access Token 발급
전달	요청에 토큰을 어떻게 싣는가	Authorization: Bearer <access_token>
검증	서버가 무엇을 확인하는가	서명·만료·역할 (복구 실행은 operator/admin만 허용)
오류	실패하면 무엇을 돌려주는가	401 토큰 누락·만료 / 403 권한 부족

401은 "누구인지 모른다", 403은 "누구인지는 알지만 권한이 없다"는 뜻이다. 둘을 구분해서 적는다.

API 목록 표로 전체를 한 장에 담는다

메서드	경로	인증	설명	관련 요구사항
GET	/api/v1/incidents	JWT	장애 목록 조회	FR-01
POST	/api/v1/incidents/{id}/diagnose	JWT	AI 진단 실행	FR-02
POST	/api/v1/incidents/{id}/recovery	JWT + 운영자	복구 작업 요청	FR-03
GET	/api/v1/recovery-jobs/{id}	JWT	복구 작업 상태 조회	FR-04

'관련 요구사항' 열로 기획서와의 연결을 보인다. 요구사항에 필요한 API만 적는다.
모든 자원의 CRUD 전체, 작업 큐, SSE는 팀이 실제로 쓰는 경우에만 넣는다.

상세 명세는 여덟 칸을 채운다

구현할 API마다 표 하나. 오늘은 핵심 1개로 초안을 쓰고, 최종 제출에는 전체를 채운다

1

호출 목적

누가, 언제, 왜 부르는가

2

요청 헤더

Authorization, X-Request-ID 등

3

경로·쿼리

이름, 타입, 필수 여부, 기본값

4

요청 본문

필드 명세 표와 실제 값의 JSON

5

검증·권한

형식 검증과 권한·승인 조건

6

정상 응답

상태 코드, 필드 명세, 예시 JSON

7

오류 응답

상태 코드별 발생 조건과 오류 코드

8

실행 규칙

타임아웃, 재시도, 중복 방지

예시: 복구 요청 API의 요청과 응답

- 202는 "접수했지만 처리는 끝나지 않았다"는 뜻이다. 실행이 시작됐거나 성공했다는 보장은 아니다.
- 결과는 job_id로 따로 조회해야 안다.
- 예시값은 다른 산출물과 같은 ID를 쓴다. (inc-001, apr-001, req-001)

```
POST /api/v1/incidents/{incident_id}/recovery

POST /api/v1/incidents/inc-001/recovery
Authorization: Bearer <JWT>
Content-Type: application/json
X-Request-ID: req-001

{
  "action": "restart",
  "target_id": "svc-api-01",
  "approval_id": "apr-001"
}

HTTP 202
{
  "job_id": "job-001",
  "status": "queued",
  "incident_id": "inc-001"
}
```

JSON 예시 옆에 필드 명세 표를 둔다

필드	위치	타입	필수	빈값	제약	설명
`incident_id`	경로	`string`	필수	불가	inc-로 시작	대상 장애
`dry_run`	쿼리	`bool`	선택	-	기본 false	모의 실행 여부
`action`	본문	`string`	필수	불가	restart, reconnect, fallback	복구 방식
`target_id`	본문	`string`	필수	불가	1~64자	대상 서비스
`approval_id`	본문	`string`	restart면 필수	불가	승인된 ID	운영자 승인
`job_id`	응답	`string`	항상	불가	job-로 시작	조회용 작업 ID
`status`	응답	`string`	항상	불가	queued	접수 상태

예시 JSON 하나로는 규칙이 보이지 않는다. 필드마다 타입·필수·빈값·제약을 적고, 예시 값과 맞춘다.
빈값은 ""(빈 문자열)이나 null을 받아도 되는지다.

오류 응답도 설계의 일부다

상태 코드	발생 조건	오류 코드(예시)
422 (또는 400)	형식이 잘못된 입력 (허용 밖 action, 길이 초과)	VALIDATION_ERROR
401	토큰 누락·만료	UNAUTHORIZED
403	역할 권한 부족	FORBIDDEN
403	승인이 없거나, 승인 전이거나, 다른 대상의 승인	APPROVAL_REQUIRED
404	장애 ID 없음	INCIDENT_NOT_FOUND
409	같은 장애에 실행 중인 복구가 있음	DUPLICATE_JOB

입력 형식은 Pydantic이 검사하고(422), 권한과 승인은 서버 로직이 검사한다(403). 두 검사를 섞지 않는다.
FastAPI는 형식 오류를 기본 422로 응답한다. 400과 공통 오류 형식을 쓰려면 예외 핸들러를 추가한다.

입력 형식은 Pydantic 모델로 검사한다

- Literal: 허용하는 값의 목록
- Field: 길이와 형식 제한
- 여기서 걸리면 FastAPI가 422로 응답한다.
- 승인이 실제로 있는지는 여기서 판단하지 않는다. 서버가 DB의 승인 기록으로 확인한다 (다음 장).

schemas.py

```
from typing import Literal
from pydantic import BaseModel, Field

class RecoveryRequest(BaseModel):
    action: Literal["restart", "reconnect", "fallback"]
    target_id: str = Field(min_length=1, max_length=64)
    approval_id: str | None = Field(
        default=None, pattern=r"^apr-[0-9a-z-]+$")

# 422: action="reboot"          (허용 값 밖)
# 422: target_id=""             (길이 제한)
# 통과: approval_id 없음  → 승인 검사는 서버에서 403
```

권한과 승인은 서버가 확인한다

- 역할: 이 사용자가 복구를 요청할 수 있는가
- 소유권: 이 사용자가 볼 수 있는 장애인가
- 승인 상태: 승인 기록이 있고, 승인됐고, 같은 장애와 대상에 대한 것인가
- 요청 본문의 approval_id는 주장일 뿐이다.

routes.py (형식 예시)

```
RISKY = {"restart"}

@router.post("/incidents/{incident_id}/recovery",
             status_code=202)
def request_recovery(incident_id: str, body: RecoveryRequest,
                     user: User = Depends(current_user)):
    if user.role not in {"operator", "admin"}:
        raise HTTPException(403, {"code": "FORBIDDEN"})
    incident = find_incident(incident_id, user)  # 없으면 404
    if body.action in RISKY:
        apr = find_approval(body.approval_id)
        if not (apr and apr.status == "approved"
                and apr.incident_id == incident.id
                and apr.target_id == body.target_id):
            raise HTTPException(
                403, {"code": "APPROVAL_REQUIRED"})
    return enqueue_job(incident, body)
```

타임아웃·재시도·중복 실행 방지 규칙을 적는다

항목	규칙 (예시 값)
타임아웃	API 응답 10초, Tool 호출 30초. 넘으면 TOOL_TIMEOUT으로 처리한다
재시도 대상	타임아웃·429·5xx 같은 일시적 오류만. 400·401·403·404·422는 재시도하지 않는다
재시도 횟수	최대 2회, 1초 → 2초 간격으로 늘린다 (지수 백오프)
중복 실행 방지	같은 Idempotency-Key로 다시 오면 처음 만든 job을 그대로 돌려준다
동시 실행	같은 장애에 실행 중인 복구가 있으면 409 DUPLICATE_JOB
변경 작업	응답이 끊겨도 이미 실행됐을 수 있다. 상태를 조회한 뒤 재시도한다

숫자는 예시다. 팀이 정한 값을 적고, Auto-Healing 구현 결과서의 재시도 규칙과 같게 맞춘다.

REST API와 Tool은 호출하는 주체가 다르다

구분	REST API	Tool (Function Calling)
호출 주체	화면(프런트엔드), 외부 시스템	에이전트(LLM)
호출 방법	HTTP 메서드 + 경로	Tool 이름 + JSON 인자
권한 확인	JWT에 담긴 사용자 역할	에이전트 허용 목록 + 요청한 사용자의 권한
문서에 적는 것	요청·응답·상태 코드	입력 스키마·출력·실패 처리·실행 조건

LLM이 만든 인자도 사용자 입력처럼 다룬다. Tool 안에서 같은 검증을 한 번 더 한다.

Tool 정의는 에이전트가 읽는 명세다

- 이름은 동사_명사: restart_service
- 호출 주체와 권한: RecoveryAgent, recovery.execute
- 실행 조건: 서버가 승인 상태를 확인한 뒤
- 출력과 실패 처리: 다음 장의 형식으로 적는다
- description은 LLM이 언제 쓸지 판단하는 근거다. 구체적으로 쓴다.

Tool 입력 스키마

```
{
  "name": "restart_service",
  "description": "승인된 대상 서비스를 재시작한다",
  "parameters": {
    "type": "object",
    "properties": {
      "target_id":   {"type": "string"},
      "approval_id": {"type": "string"},
      "request_id":  {"type": "string"}
    },
    "required": ["target_id", "approval_id", "request_id"]
  }
}
```

Tool의 성공·실패 출력과 실패 후 처리

- 성공과 실패의 형식을 하나로 맞춘다 (ok 필드)
- TOOL_TIMEOUT: 실제로 실행됐는지 조회한 뒤 1회 재시도
- TARGET_NOT_FOUND: 재시도하지 않고 진단 단계로 되돌린다
- APPROVAL_REQUIRED: 실행을 멈추고 승인을 요청한다
- retryable로 에이전트가 재시도 여부를 판단한다

restart_service 출력 (형식 예시)

```
// 성공
{"ok": true,
 "data": {"job_id": "job-001", "status": "started"},
 "request_id": "req-001"}

// 실패
{"ok": false,
 "error": {"code": "TOOL_TIMEOUT",
           "message": "30초 안에 응답 없음",
           "retryable": true},
 "request_id": "req-001"}
```

Tool 실행 전에 서버가 네 가지를 확인한다

① 에이전트 허용 목록

허용 목록은 첫 번째 관문일 뿐이다.

```
RecoveryAgent → restart_service
```

② 요청한 사용자의 권한

사용자가 못 하는 일은 에이전트도 못 한다.

```
user.role in {operator, admin}
```

③ 데이터 소유권

대상이 그 사용자·조직의 자원인가.

```
target_id in user.assets
```

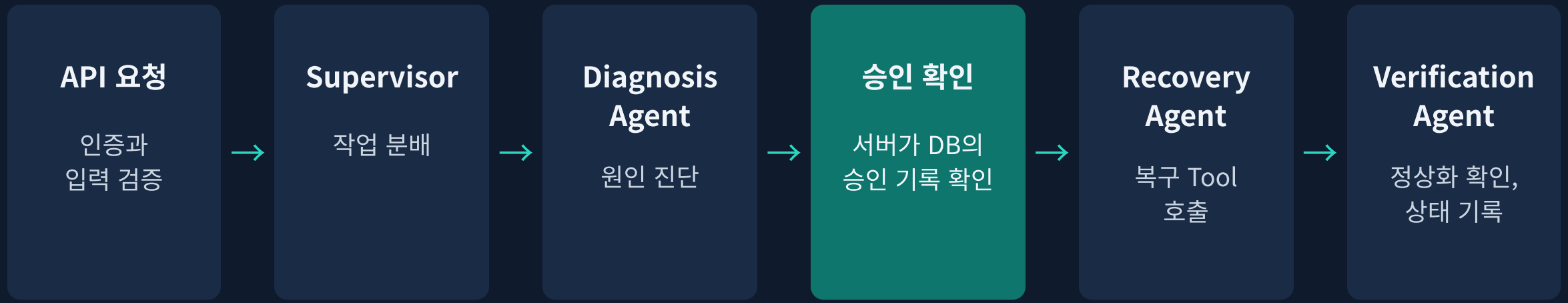
④ 실제 승인 상태

LLM이 넘긴 승인 ID를 그대로 믿지 않는다.

```
approval.status == "approved"
```

Tool 호출 흐름을 한 줄로 그린다

요청 하나가 어떤 순서로 에이전트와 Tool을 거치는지 문서에 남긴다



API의 X-Request-ID와 Tool의 request_id를 같은 값으로 이어 두면, 로그에서 요청 전체를 한 번에 추적할 수 있다.

응답 방식은 작업 시간에 맞춰 고른다

모든 에이전트 API를 비동기로 만들 필요는 없다. 팀이 실제로 쓰는 방식만 문서에 적는다

① 바로 응답

대부분은 200으로 끝낸다

몇 초 안에 끝나는 조회와 처리는 결과를 바로 돌려준다.

```
GET /incidents/{id}
→ 200 {"id": "inc-001"}
```

② 접수와 조회

오래 걸리면 202

접수만 하고 job_id를 돌려준다. 결과는 조회로 확인한다.

```
POST …/recovery → 202
GET /recovery-jobs/{id}
```

③ 스트리밍 (쓰는 팀만)

진행 상황을 흘려보낸다

실시간 화면을 만드는 팀만 이벤트 종류를 명세한다.

```
POST /api/v1/chat/stream
→ text/event-stream
```

작업 조회 응답은 상태·결과·실패를 구분한다

- 상태 값: queued → running → succeeded 또는 failed
- 작업이 실패해도 조회 요청은 성공했으므로 200이다. 실패는 status와 error로 알린다.
- 결과는 succeeded일 때만 result에 담는다.
- 없는 job_id는 404다.

```
GET /api/v1/recovery-jobs/{job_id} (형식 예시)

GET /api/v1/recovery-jobs/job-001

// 처리 중
200 {"job_id": "job-001", "status": "running"}

// 성공
200 {"job_id": "job-001", "status": "succeeded",
     "result": {"health": "ok"},
     "finished_at": "2026-11-18T10:00:42+09:00"}

// 실패
200 {"job_id": "job-001", "status": "failed",
     "error": {"code": "TOOL_TIMEOUT",
               "message": "30초 안에 응답 없음"}}
```

다른 산출물과 연결한다

문서마다 이름과 규칙이 따로 놀면 설계·구현·검증의 연결을 보여줄 수 없다

기획서

요구사항 ID를 연결한다

API마다 FR-01 같은 요구사항 번호를 붙인다.

DB 스키마 설계서

필드와 컬럼의 대응을 밝힌다

이름은 달라도 된다. 어느 필드가 어느 컬럼에 저장되는지 분명하면 된다. ID 형식은 같게 쓴다.

AI 보안 가드레일 문서

권한 규칙이 일치한다

역할별·Tool별 권한표와 403 조건이 서로 같아야 한다.

통합 테스트 결과서

오류 표가 테스트가 된다

"승인 없는 요청 → 403" 한 줄이 테스트 케이스 하나다.

자주 나오는 실수와 고치는 법

실수	이렇게 고친다
같은 API인데 문서 안에서 경로나 필드명이 다르다 (/recover 와 /recovery, action 과 strategy)	API 목록 표를 기준으로 삼고 상세 명세를 거기에 맞춘다
JSON 예시만 있고 필드 규칙이 없다	필드 명세 표(타입·필수·빈값·제약)를 추가한다
승인 없음을 Pydantic 오류(422)로 처리했다	형식 검증과 서버의 승인 검사(403)를 나눈다
Tool 허용 목록만 있고 서버 권한 검사가 없다	사용자 권한·소유권·승인 상태를 서버가 확인한다
구현하면서 문서와 코드가 달라졌다	FastAPI의 /docs 화면과 대조하고 문서 버전을 올린다

우리 서비스의 대상과 행위로 바꿔 쓴다

아래는 실제 팀 주제와 관계없는 일반 가상 예시다. 형태만 참고한다

가상 예시 ① 회의실 예약 비서

대상: 회의실, 예약

같은 시간 중복 예약을 막는 409가 핵심이다.

```
GET /rooms?date=…
POST /reservations
Tool: check_availability
```

가상 예시 ② 도서 추천 상담

대상: 도서, 상담

검색 조건을 쿼리 파라미터로 명세한다.

```
GET /books?keyword=…
POST /consultations
Tool: search_books
```

가상 예시 ③ 배송 지연 알림

대상: 배송, 알림

외부 택배 API를 부르는 Tool의 실패 처리가 핵심이다.

```
GET /shipments/{id}
POST /shipments/{id}/notify
Tool: get_tracking
```

최종 제출 전 체크리스트

- [ ] 구현할 API가 모두 목록 표에 있고 요구사항 ID와 연결된다
- [ ] 202를 쓰는 API는 작업 상태·결과·실패 조회가 있다
- [ ] 서버 주소·인증·오류 형식·/docs·변경 이력이 있다
- [ ] Tool마다 성공·실패 출력과 실패 후 처리가 있다
- [ ] 구현할 API마다 상세 명세와 필드 명세 표가 있다
- [ ] 타임아웃·재시도·중복 실행 방지 규칙이 있다
- [ ] 형식 오류(422)와 권한·승인 오류(403)를 나눠 적었다
- [ ] ID 형식이 같고, 필드와 DB 컬럼의 대응이 분명하다

오늘 팀에서 할 일

완벽한 문서보다 합의된 초안이 먼저다. 구현하면서 버전을 올린다

① API 목록 표

요구사항 전체를
표 한 장으로

→

② 상세 명세 1개

핵심 API 하나로
초안 쓰기

→

③ Tool 정의 1개

핵심 Tool 하나로
초안 쓰기

→

④ 팀 리뷰

프런트·백엔드·AI
담당이 함께 읽기

②·③은 오늘의 초안 실습 범위다. 최종 제출에는 구현하기로 확정한 API·Tool 전체의 명세가 있어야 한다.
문서 머리말에 문서 버전과 작성 상태를 적는다. 예: 문서 버전 v0.1 | 작성 상태: 초안