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 식별자와 필드명을 써야 한다.

문서에 들어가는 여섯 가지

01

인증 방식

토큰 발급·전달·검증 방법과 401·403 조건

02

API 목록

메서드·경로·인증·설명을 한눈에 보는 표

03

엔드포인트 상세 명세

요청·응답·오류를 예시값과 함께

04

입력 검증 규칙

허용 값·길이·필수 조건을 Pydantic 모델로

05

Tool 정의

이름·호출 주체·입력·출력·실패 코드

06

Tool 호출 흐름

요청이 에이전트와 Tool을 거치는 순서

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
```

RESTful하지 않은 주소를 고쳐 쓴다

이렇게 쓰지 않는다	이렇게 쓴다	이유
`GET /getIncidents`	`GET /incidents`	동사는 메서드가 이미 말하고 있다
`GET /incident/list`	`GET /incidents`	목록은 복수형 명사 하나로 충분하다
`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

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.

주소와 메서드를 정하는 규칙

주소는 명사, 복수형으로

동사는 주소에 넣지 않고 메서드에 담는다.

```
/incidents    /recovery-jobs
```

메서드가 곧 행위다

GET 조회, POST 생성·실행, PATCH 수정, DELETE 삭제

```
GET · POST · PATCH · DELETE
```

버전을 앞에 붙인다

나중에 규격을 바꿔도 기존 화면이 깨지지 않는다.

```
/api/v1/…
```

포함 관계는 경로로 표현한다

특정 장애에 대한 복구 요청은 그 장애 아래에 둔다.

```
/incidents/{id}/recovery
```

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' 열을 더하면 기획서와의 연결이 바로 보인다. 구현한 API는 반드시 목록에 있고, 목록에 없는 API는 만들지 않는다.

상세 명세는 여덟 칸을 채운다

핵심 엔드포인트마다 표 하나. 빈칸이 남아 있으면 설계가 아직 끝나지 않은 것이다

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

실제 값이 들어간 JSON

5

입력 검증

허용 값, 길이, 조건부 필수

6

정상 응답

상태 코드와 응답 JSON

7

오류 응답

상태 코드별 발생 조건

8

응답 예시

오류 JSON의 형식

예시: 복구 요청 API의 요청과 응답

- 202는 "접수했고 처리 중"이라는 뜻이다. 오래 걸리는 작업에 쓴다.
- 응답의 job_id로 진행 상태를 따로 조회한다.
- 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"
}
```

오류 응답도 설계의 일부다

정상 응답만 적은 문서는 절반짜리다. 실패 조건과 오류 코드를 표로 만든다

상태 코드	발생 조건	오류 코드(예시)
400	허용되지 않는 값	INVALID_ACTION
401	토큰 누락·만료	UNAUTHORIZED
403	권한 또는 승인 없음	APPROVAL_REQUIRED
404	장애 ID 없음	INCIDENT_NOT_FOUND
409	같은 복구가 이미 실행 중	DUPLICATE_JOB

오류 형식은 하나로 통일한다: {"error": {"code": "APPROVAL_REQUIRED", "message": "승인 필요"}}
FastAPI는 입력 검증 실패를 기본 422로 돌려준다. 400과 422 중 무엇을 쓸지 팀에서 정해 문서에 적는다.

입력 검증은 Pydantic 모델로 적는다

- Literal: 허용하는 값의 목록
- Field: 길이와 범위 제한
- model_validator: 여러 필드가 얽힌 조건. 위험한 작업이면 승인 ID가 필수
- 문서의 검증 규칙과 코드의 모델은 같아야 한다.

schemas.py

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

RISKY = {"restart"}

class RecoveryRequest(BaseModel):
    action: Literal["restart", "reconnect", "fallback"]
    target_id: str = Field(min_length=1, max_length=64)
    approval_id: str | None = None

    @model_validator(mode="after")
    def need_approval(self):
        if self.action in RISKY and not self.approval_id:
            raise ValueError("APPROVAL_REQUIRED")
        return self
```

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	에이전트별 허용 Tool 목록
문서에 적는 것	요청·응답·상태 코드	입력 스키마·출력·실패 코드·실행 조건

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

Tool 정의는 에이전트가 읽는 명세다

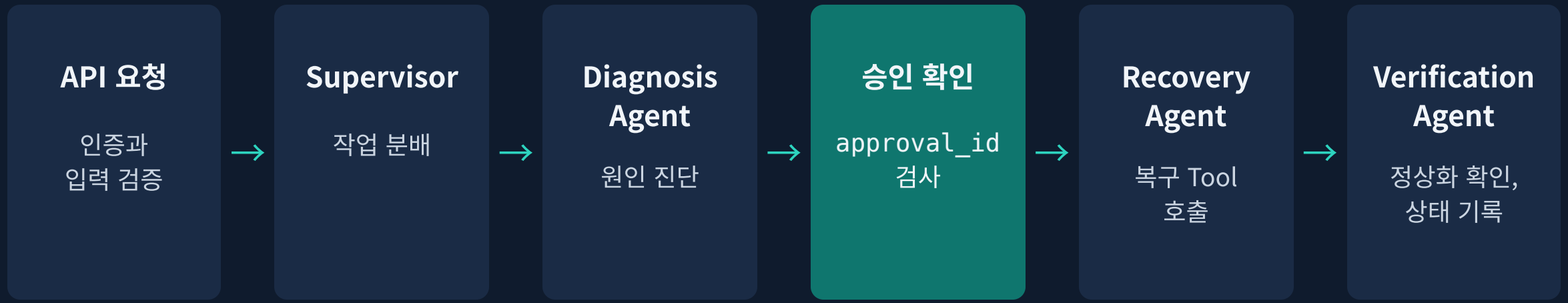
- 이름은 동사_명사: restart_service
- 호출 주체와 권한: RecoveryAgent, recovery.execute
- 실행 조건: 운영자 승인을 확인한 뒤
- 출력과 실패 코드: job_id, status / TOOL_TIMEOUT, TARGET_NOT_FOUND, APPROVAL_REQUIRED
- description은 LLM이 언제 쓸지 판단하는 근거다. 구체적으로 쓴다.

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"]
  }
}
```

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.

오래 걸리는 작업은 접수와 조회로 나눈다

에이전트 실행은 몇 초에서 몇 분이 걸린다. 응답을 붙잡고 기다리게 하지 않는다

① 접수

바로 202를 돌려준다

작업을 대기열에 넣고 job_id를 즉시 반환한다.

```
POST …/recovery
→ 202 {"job_id": "job-001"}
```

② 조회

상태를 따로 묻는다

가능한 상태 값의 목록을 문서에 적어 둔다.

```
GET /recovery-jobs/{id}
→ {"status": "queued"}
```

③ 스트리밍

진행 상황을 흘려보낸다

STEP 5의 실시간 화면에서 쓴다. 이벤트 종류를 명세한다.

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

다른 산출물과 같은 이름을 쓴다

문서마다 이름이 다르면 설계·구현·검증의 연결을 보여줄 수 없다

기획서

요구사항 ID를 연결한다

API마다 FR-01 같은 요구사항 번호를 붙인다.

DB 스키마 설계서

필드명과 컬럼명을 맞춘다

incident_id, approval_id를 두 문서에서 똑같이 쓴다.

AI 보안 가드레일 문서

권한 규칙이 일치한다

역할별·Tool별 권한표와 인증 규칙이 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
정상 응답만 적었다	상태 코드별 오류 표를 추가한다
"문자열", "숫자"만 있고 예시값이 없다	실제 값이 들어간 JSON을 넣는다
Tool에 실패 코드와 실행 조건이 없다	실패했을 때 에이전트가 할 행동까지 적는다
구현하면서 문서와 코드가 달라졌다	FastAPI의 /docs 화면과 대조하고 문서 버전을 올린다

팀 주제로 바꿔 쓴다

샘플의 '장애·복구'를 우리 서비스의 대상과 행위로 바꾼다. 아래는 형태만 보여주는 예시다

묶음배달 오토힐링

대상: 배달, 장애

복구 요청과 상태 조회 구조를 그대로 쓸 수 있다.

```
GET /deliveries
POST /incidents/{id}/heal
Tool: reassign_rider
```

축제 안심 관광 비서

대상: 행사, 혼잡도

외부 공공 API를 부르는 Tool의 실패 코드가 핵심이다.

```
GET /events/{id}/congestion
POST /chat/stream
Tool: get_crowd_level
```

자전거 상담·코스 추천

대상: 코스, 상담

검색 조건을 쿼리 파라미터로 명세한다.

```
GET /courses?region=…
POST /consultations
Tool: search_safety_info
```

제출 전 체크리스트

- [ ] 모든 API가 목록 표에 있고 요구사항 ID와 연결된다
- [ ] 상태 코드별 오류 조건과 오류 코드 표가 있다
- [ ] 인증 방식과 401·403 조건이 적혀 있다
- [ ] 검증 규칙이 Pydantic 모델과 일치한다
- [ ] 핵심 엔드포인트의 상세 명세 여덟 칸이 모두 채워졌다
- [ ] Tool마다 입력 스키마·출력·실패 코드·실행 조건이 있다
- [ ] 요청과 응답에 실제 값이 든 JSON 예시가 있다
- [ ] 경로·필드명·ID가 DB 스키마 등 다른 산출물과 같다

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 엔드포인트의
여덟 칸 채우기

→

③ Tool 정의 1개

입력 스키마와
실패 코드까지

→

④ 팀 리뷰

프런트·백엔드·AI
담당이 함께 읽기

문서 머리말에 문서 버전과 작성 상태를 적는다. 예: 문서 버전 v0.1 | 작성 상태: 초안