

Module 2 · Chapter 1 — 명세서를 실행 가능한 시스템으로 옮기기

Confluence 공간 2개 · Jira 프로젝트 7개를 명세서에서 실제 Atlassian 사이트까지 옮긴 과정. **AI가 만든 명세와 실제 API** 사이의 간극을 어디서, 어떻게 발견하고 흡수했는가가 이 챕터의 주제입니다.

모듈

Module 2 · Chapter 1

난이도

실무

학습 시간

70분

선행 학습

Module 1 (R-C-T-C-F 프롬프팅 기초)

학습 목표 (Learning Objectives)

이 챕터를 마치면 수강자는 다음을 할 수 있습니다.

- ✓ AI가 생성한 기획 명세를 외부 **API** 제약과 대조해 실행 가능한 형태로 정규화할 수 있다
- ✓ 문서가 아니라 **설정 파일(JSON)**을 단일 원본으로 두는 구조를 설계할 수 있다
- ✓ 재실행해도 안전한 **멥등(idempotent)** 부트스트랩과 **DRY-RUN** 기본값을 구현할 수 있다
- ✓ 자동화할 수 있는 것과 수동으로 남겨야 하는 것을 API 지원 범위로 판별할 수 있다
- ✓ 명세 표기와 실제 시스템 값이 달라질 때 **추적성을 잃지 않는** 방법을 적용할 수 있다

실무 배경 — 어떤 문제에서 출발했나

- **상황:** IT PM/PO 업무 자동화 4-Product 시스템을 만들기로 하고, Google Docs에 프로덕트 4개의 명세서를 작성했습니다. 여기에는 Confluence 공간 구조와 Jira 프로젝트 KEY가 구체적으로 적혀 있었습니다.

```
Product 1: 'PM/PO AI 기획 웹' (KEY: IDEA_WEB)
Product 3: 'PM/PO AI 강의 웹' (KEY: LECTURE_WEB)
Product 2: TOOL_A / TOOL_B / TOOL_C / TOOL_D
Confluence 공간: PMPO_WIKI, PMPO_TOOLS
```

- **기존 방식의 비용:** 공간 2개 + 페이지 22개 + 프로젝트 7개 + Epic 20여 개 + 백로그 40여 건을 UI에서 손으로 만들면 반나절이 걸립니다. 더 큰 비용은 구조가 바뀔 때마다 전부 다시 만져야 한다는 점입니다.
- **진짜 문제:** 명세서에 적힌 KEY 중 **7개 중 6개가 Atlassian API에서 거부되는 값**이었습니다. 하지만 그걸 아무도 몰랐습니다 — 명세서는 그럴듯했고, 검토도 통과했습니다.

실무 배경 — 어떤 문제에서 출발했나 (계속)

💡 이 챕터의 핵심 교훈: AI가 쓴 기획서는 *읽기에* 완벽할 수 있습니다.

그것이 *실행 가능한지*는 완전히 다른 문제이고, 실행해 보기 전에는 알 수 없습니다.

3.1 프롬프팅 프레임워크 적용 — R-C-T-C-F

실제로 사용한 프롬프트의 구조입니다.

요소	실제 입력값
Role	Full-Stack Software Engineer & AI Automation Architect
Context	IT PM/PO 업무 자동화 4-Product 시스템. Confluence·Jira를 단일 원본으로 관리. 계정 비밀번호는 제공하지 않고 API 토큰만 사용
Task	Confluence 공간/페이지 트리 세팅 가이드 + Jira 프로젝트·백로그 세팅 자동화 스크립트/명세서 작성
Constraints	먹통 실행, DRY-RUN 기본값, 비밀값 커밋 금지, 명세와 API 규칙이 충돌하면 먼저 알리고 정규화한 값으로 진행
Format	Markdown 가이드 + Python CLI + 스펙 JSON

Constraints 한 줄이 결과를 갈랐습니다.

3.1 프롬프팅 프레임워크 적용 — R-C-T-C-F (계속)

"명세와 API 규칙이 충돌하면 먼저 알려주고 정규화한 값으로 진행한다"

이 문장이 없었다면 AI는 두 가지 중 하나를 했을 겁니다 — ①명세대로 `IDEA_WEB` 을 써서 실행 시점에 400 에러로 죽거나, ②말없이 `IDEAWEB` 으로 바꿔서 나중에 명세서와 실제 시스템이 어긋난 걸 아무도 모르거나. 둘 다 나쁩니다.

 **실습 포인트:** 프롬프트에 "충돌하면 먼저 알려라"를 넣는 것과 넣지 않는 것의 차이를 직접 비교해 보세요. 이것이 Module 1에서 배운 Constraints의 실전 효과입니다.

3.2 아키텍처 — 문서가 아니라 JSON이 단일 원본



docs/4-product-system/*.md ← JSON을 '설명'만 함. 여기를 고쳐도 아무것도 안 바뀜

3.2 아키텍처 — 문서가 아니라 JSON이 단일 원본 (계속)

왜 이 구조인가: 문서와 코드가 각자 구조를 알고 있으면 반드시 어긋납니다. JSON 하나만 진실로 두고, 문서는 그것을 설명하고, 코드는 그것을 실행합니다. 실제로 이 프로젝트에서 구조가 **3번 바뀌었는데**(KEY 명명 규칙이 두 번, 공간 키가 한 번) 매번 JSON 한 파일만 고치고 스크립트를 재실행해 끝났습니다.

구현 과정 (Step by Step)

단계	작업	소요	산출물
1	명세서·대시보드 원문 읽고 실제 Atlassian 상태 조회	10분	현황 파악
2	KEY 규칙 위반 발견 및 정규화 설계	15분	매핑 표
3	스펙 JSON 2종 작성 (공간·페이지 / 프로젝트·백로그)	40분	<code>config/specs/</code>
4	먹등 클라이언트 + 부트스트랩 구현	90분	<code>common/</code> , <code>setup_ops/</code>
5	표준 템플릿 4종 + 세팅 가이드 3종 작성	60분	<code>docs/4-product-system/</code>
6	테스트 작성 및 DRY-RUN 검증	40분	pytest 86건

§4

4.1 KEY 규칙 위반 — 가장 중요한 15분

Atlassian의 실제 규칙:

대상	규칙	명세서 값	결과
Jira 프로젝트 KEY	<code>^[A-Z][A-Z0-9]{1,9}\$</code>	IDEA_WEB	✗ 언더스코어 불가
Confluence 공간 KEY	<code>^[A-Za-z0-9]+\$</code>	PMP0_WIKI	✗ 언더스코어 불가

정규화하면서 **추적성을 잃지 않는 것**이 설계의 핵심이었습니다.

4.1 KEY 규칙 위반 — 가장 중요한 15분 (계속)

명세 표기	실제 KEY	보존 방법
IDEA_WEB	IDEAWEB	JSON <code>requestedKey</code> + Jira 라벨 <code>spec:IDEA_WEB</code>
LECTURE_WEB	LECTWEB	//
TOOL_A ~ TOOL_D	TOOLA ~ TOOLD	//
PMPO_WIKI	PMPOWIKI	JSON <code>requestedKey</code>
PMPO_TOOLS	PMPOTTOOLS	//

4.1 KEY 규칙 위반 — 가장 중요한 15분 (계속)

```
def normalize_key(raw: str) -> str:
    """'DATA_PIPE' → 'DATAPIPE'. 규칙 위반이면 ValueError."""
    key = re.sub(r"^[A-Za-z0-9]", "", raw).upper()[:10]
    if not KEY_PATTERN.match(key):
        raise ValueError(
            f"Jira 프로젝트 KEY 로 쓸 수 없습니다: {raw!r} → {key!r}. "
            "규칙: 대문자로 시작, 대문자·숫자만, 2~10자."
        )
    return key
```

그리고 실행 전에 차단하는 명령을 따로 만들었습니다. 400 에러를 20분 뒤에 만나는 것보다 0초에 표로 보는 게 낫습니다.

4.1 KEY 규칙 위반 — 가장 중요한 15분 (계속)

```
$ python main.py setup validate
```

KEY 검증

종류	명세 표기	실제 생성 KEY	결과
Confluence 공간	PMPO_WIKI	PMPOWIKI	정규화된
Jira 프로젝트	IDEA_WEB	IDEAWEB	정규화된
Jira 프로젝트	TOOL_A	TOOLA	정규화된
Jira 프로젝트	PORTFOLIO	PORTFOLIO	OK

4.2 멍등성 — 재실행이 안전해야 손이 가벼워진다

부트스트랩은 몇 번을 돌려도 같은 결과여야 합니다. 그래야 "일단 돌려보자"가 가능합니다.

```
def ensure_project(self, *, key, name, description="", ...):
    """프로젝트가 없으면 생성하고, 있으면 그대로 반환합니다."""
    key = normalize_key(key)
    if existing := self.find_project(key):
        logger.info("Jira 프로젝트 이미 존재: %s (%s)", key, existing.get("name"))
        return existing
    ...
```

이슈는 `project + summary` 완전일치로 조회한 뒤 없을 때만 만듭니다. Confluence 페이지는 제목 기준 upsert이되 기존 본문은 덮어쓰지 않습니다 — 사람이 채운 위키 내용을 스크립트 재실행이 날리는 사고를 막기 위한 기본값입니다.

4.3 DRY-RUN이 기본값 — 쓰기는 막고 읽기는 통과

```
write = method.upper() in {"POST", "PUT", "DELETE"} and not read_only
if write and self.dry_run:
    logger.info("[DRY-RUN] %s %s %s", method, url, _preview(json))
    return dry_run_result if dry_run_result is not None else {}
```

읽기는 **dry-run**에서도 실제로 보냅니다. 그래야 "이미 있는 것 / 새로 만들 것"을 정확히 구분해 보여줄 수 있습니다. 읽기까지 막으면 DRY-RUN 출력이 거짓말을 하게 됩니다.

4.4 자동화할 수 없는 것을 정직하게 남기기

전부 자동화하려다 실패하는 것보다, 경계를 명확히 긋고 수동 절차를 체크리스트로 주는 것이 낫습니다. API 지원 범위를 조사해 아래를 문서에 명시했습니다.

항목	왜 수동인가	대응
보드 컬럼 개명	팀 관리형 프로젝트 상태를 다루는 공개 API 없음	UI 1회 작업 + 체크리스트
Confluence 공간 템플릿 등록	REST 지원 제한적	99. 표준 템플릿 보관함 페이지로 대체
Google 서비스 계정·OAuth	콘솔 동의 필요	절차 문서화
Apps Script 설치	계정 권한 동의 필요	setupTriggers 원클릭 함수 제공

그리고 자동화 실패가 전체를 죽이지 않게 했습니다. 상태 전이가 불가능하면 경고만 남기고 이슈 생성은 계속합니다.

결과 & 검증 (Result & Validation)

지표	Before	After	측정 근거
초기 세팅 소요	반나절(수기)	약 3분(CLI) + 수동 2건	실행 로그
구조 변경 반영	전 항목 수기 재작업	JSON 1파일 수정 + 재실행	실제 3회 변경 수행
잘못된 KEY로 인한 실패	발견 시점 = 실행 중	실행 전 <code>setup validate</code> 에서 차단	exit 1
명세 ↔ 시스템 추적	정규화 시 단절	<code>requestedKey</code> + <code>spec:</code> 라벨로 유지	JQL 조회 가능

실제 생성물 (2026-08-08 실측)

Confluence

PMP0WIKI

9 페이지 (1단계 5 + Module 1~4)

PMP0T00LS

13 페이지 (Tool A~D 4 + 각 하위 2 + 템플릿 1)

Jira

프로젝트 7개 + Epic 20여 개 + Story/Task 40여 건

배운 것 / 다시 한다면 바꿀 것

-  **잘한 것:** 스펙을 JSON으로 분리한 것. 요구사항이 3번 바뀌는 동안 매번 한 파일만 고쳤습니다.
-  **잘한 것:** `setup validate` 를 만든 것. 실패를 실행 전으로 당겼습니다.
-  **다시 한다면:** 문서에 페이지 수를 "18개"라고 적었는데 실제 규격은 22개였습니다. 사람이 센 숫자를 문서에 박지 말고 스크립트가 출력하게 해야 합니다.
-  **다시 한다면:** "In Review 컬럼을 수동으로 추가해야 한다"고 문서에 썼는데, 실제로는 Atlassian 기본 템플릿이 이미 4열로 생성되었습니다(2026-08 실측). 조사하지 않고 추측으로 쓴 제약은 거짓 정보입니다.

💡 위 두 개의  는 우연이 아닙니다. 둘 다 "실행해 보지 않고 문서에 쓴 것" 입니다.

Chapter 2가 정확히 이 지점에서 출발합니다.

산출물 링크 (Deliverables)

구분	URL
교안 PDF/슬라이드	/downloads/module2-ch1-slides.pdf
슬라이드 (HTML)	/downloads/module2-ch1-slides.html
실습용 데이터 세트	<code>pm-po-automation-tools/config/specs/*.json</code>
시뮬레이터 실행	(해당 없음)
GitHub 저장소	https://github.com/ursulax/action
세팅 가이드	<code>docs/4-product-system/01-confluence-space-guide.md</code>
Jira 명세서	<code>docs/4-product-system/02-jira-project-spec.md</code>
실제 Confluence	https://awesomest.atlassian.net/wiki/spaces/PMPOWIKI
관련 PR	https://github.com/ursulax/action/pull/20

실습 과제 (Action Item)

1. 본인 업무의 반복 세팅 하나를 골라 스펙 **JSON**으로 표현하세요. (예: 프로젝트 킥오프 시 만드는 문서 5종, 매주 만드는 리포트 구조)

1. **R-C-T-C-F** 프롬프트를 작성하되, **Constraints**에 반드시 다음을 넣으세요. `` 명세와 실제 API/시스템 규칙이 충돌하면 먼저 알려주고, 정규화한 값으로 진행하되 원 표기를 추적 가능하게 보존하라. ``

1. **validate** 성격의 사전 검증 단계를 먼저 만드세요. 본 작업보다 먼저.

제출물: 스펙 JSON 1개 + 검증 함수 1개 + 정규화 매핑 표 자가 점검 기준:

- ✓ 스크립트를 두 번 돌려도 결과가 같은가 (먹등)
- ✓ DRY-RUN으로 무엇이 만들어질지 미리 볼 수 있는가
- ✓ 자동화 못 하는 항목이 체크리스트로 남아 있는가

Q&A / 자주 막히는 지점

증상	원인	해결
Jira 프로젝트 생성 시 400	KEY에 언더스코어/소문자/11자 이상	<code>normalize_key()</code> 통과 후 사용. <code>setup validate</code> 로 사전 차단
Confluence 공간 생성 시 거부	공간 KEY에 특수문자	영숫자만 사용. 표시명에는 슬래시 등 가능
MCP 연동으로 자동화가 안 됨	세션 스코프에 공간 생성·Jira 권한 없음	API 토큰 기반 REST 사용. 스코프를 먼저 확인할 것
재실행하니 티켓이 중복 생성됨	존재 확인 없이 create만 호출	<code>ensure_*</code> 패턴 — 조회 후 없을 때만 생성
사람이 채운 위키가 날아감	upsert가 항상 덮어씀	기본값을 본문 보존으로. 덮어쓰기는 명시적 옵션으로 분리
DRY-RUN 출력이 실제와 다름	읽기 호출까지 막아서 상태를 모름	쓰기만 막고 읽기는 통과시킬 것