바이브 메이커 · 케이스스터디 무료 전자책

클로드코드 릴스 자동화 제작법

클로드코드로 웹서비스 + 무인 콘텐츠 공장을 만든 전 과정

클로드코드 · goal/loop 개발법 · 워크플로우 & 하네스 · 콘텐츠·마케팅 자동화



바이브 메이커 Vibe Maker

@vibemak2r

CONTENTS

목차

01 서문 — 이 책이 보여줄 것

02 CH1 · 클로드코드가 뭔가요?

03 CH2 — 워크플로우 vs 하네스: 레시피와 주방

04 CH3 — 사주 웹서비스는 어떻게 만들었나 (goal/loop 개발법)

05 CH4 — 콘텐츠 자동화 = 영상 공장

06 CH5 — 마케팅 자동화, 원리와 노림수

07 CH6 — 무인화 = 3중 하네스: 자는 동안 돌아가는 공장

08 CH7 — 비법 정리 + 적용 로드맵

서문 — 이 책이 보여줄 것

"코드 한 줄 몰라도 서비스를 만들 수 있다"는 말은 이제 많이 들어보셨을 겁니다. 그런데 정말 **처음부터 끝까지**, 그러니까 **아이디어 → 진짜 돌아가는 웹서비스 → 손님을 데려오는 마케팅 → 자면서도 콘텐츠를 뽑아내는 자동화**까지 한 사람이 해낼 수 있을까요?

이 책은 그 질문에 대한 **실제 기록**입니다. 소재는 실존하는 서비스 하나 — **사주시바**(sajushiba)입니다. 갓 쓴 도사 시바견이 사주를 풀어주는 운세 서비스인데, 이 하나 안에 우리가 이야기할 모든 것이 들어 있습니다.

- **웹서비스**: 생년월일을 넣으면 사주팔자를 계산하고 AI가 풀어주는 실제 사이트. 결제까지 붙어 있습니다. 그리고 이걸 **하루** 만에 만들었습니다.
- **콘텐츠 자동화**: 한 줄 아이디어를 넣으면 대본·화면·목소리·검수·발행까지 흘러가는 "영상 공장". 사람은 딱 한 번만 개입합니다.
- **마케팅 자동화**: 그렇게 만든 영상을 5개 SNS로 동시에 퍼뜨려, 손님을 서비스로 데려오려는 퍼널.
- **무인화**: 이 공장을 스케줄러에 걸어두면, 사람도 AI도 없이 매일 알아서 콘텐츠가 올라갑니다.

이 모든 걸 가능하게 한 도구가 **클로드코드**이고, 그걸 다루는 방식이 이 책의 진짜 주제입니다.

이 책을 읽는 법

순서대로 읽으시면 됩니다. 뒤로 갈수록 앞의 어휘를 그대로 씁니다.

1. **개념 토대** — 클로드코드가 뭔지, 스킬·커맨드·MCP·하네스 같은 단어의 뜻 (1·2장)
2. **웹서비스를 만든 법** — goal/loop 개발법으로 하루 만에 (3장)
3. **콘텐츠·마케팅 자동화** — 영상 공장과 그 뒤의 노림수 (4·5장)
4. **무인화** — 자면서 도는 3중 하네스 (6장)
5. **비법 정리** — 당신에게 그대로 이식하는 법 (7장)

> ⚙️ **더 깊이** — 파일 경로·수치·보안 같은 기술 디테일은 이렇게 생긴 박스에 따로 담았습니다. 비전공자라면 처음엔 건너뛰고 본문만 따라가도 완독됩니다. "더 알고 싶다"는 갈증이 생겼을 때 펼치세요.

한 가지 정직하게

이 책은 "돈을 이만큼 벌었다"는 성공담이 **아닙니다**. 마케팅 자동화가 실제로 얼마나 손님을 데려오는지(전환율·획득비용 같은 숫자)는 **아직 검증 중**입니다. 이 자동화는 "이미 성과를 낸 기계"가 아니라 "**그 데이터를 만들어 검증하려는 베팅**"에 가깝습니다.

그래서 이 책의 진짜 가치는 결과가 아니라 **만드는 방식**에 있습니다. 같은 방식이면 사주가 아니라 당신의 아이디어에도 그대로 적용됩니다. 그 방식을 처음부터 끝까지, 숨김없이 보여드리겠습니다.

CH1 · 클로드코드가 뭔가요?

혹시 이런 경험 있으신가요. 챗봇에게 "이런 웹사이트 만들어 줘"라고 부탁했더니, 답으로 코드 한 뭉치가 주르륵 나옵니다. 그런데 그걸 어디에 붙여넣어야 하는지, 파일 이름은 뭐로 해야 하는지, 저장은 어떻게 하는지는 아무도 안 알려줍니다. 결국 그 코드를 복사해서 어딘가에 옮기고, 에러가 나면 다시 물어보고, 또 복사하고... 이 복붙 노동을 몇 시간 하다 보면 "이럴 거면 그냥 사람을 쓰지" 소리가 절로 나옵니다.

이 책의 주인공인 **사주시바**라는 서비스는, 바로 그 복붙 노동을 거의 하지 않고 만들어졌습니다. 운세를 봐 주는 웹서비스 하나, 그리고 그 서비스를 홍보하는 숏폼 영상을 매일 자동으로 찍어내는 콘텐츠 공장 하나. 이 두 가지를 만든 도구가 **클로드 코드(Claude Code)**입니다.

이 첫 챕터의 목표는 딱 하나입니다. **앞으로 이 책이 계속 쓸 단어들을 미리 익혀 두는 것.** 스킬, 커맨드, MCP, 서브에이전트... 이름만 들으면 어질어질하지만, 하나씩 뜯어보면 전부 우리가 이미 아는 개념들입니다. 신입사원, 매뉴얼, 도구상자, 심부름. 이 챕터를 다 읽고 나면, 뒤 챕터에서 "이건 스킬로 처리했어요"라는 문장이 나와도 "아, 그 매뉴얼 얘기구나" 하고 자연스럽게 넘어가실 수 있습니다. 그게 목표입니다.

챗봇과 클로드코드는 뭐가 다른가요

가장 먼저, 가장 중요한 구분부터 하겠습니다. 우리가 흔히 쓰는 **챗봇**(예: 화면에 대화창이 뜨는 AI)과 **클로드코드**는 근본적으로 하는 일이 다릅니다.

챗봇은 "대화하고 답을 주는" AI입니다. 제가 질문하면 답을 텍스트로 돌려줍니다. 코드를 달라고 하면 코드를 글자로 보여줍니다. 하지만 거기까지입니다. 그 코드를 내 컴퓨터의 어느 폴더에 넣고, 어떤 파일로 저장하고, 어떻게 실행할지는 **전부 내 몫**입니다. 챗봇은 내 컴퓨터 안을 볼 수 없거든요. 마치 요리 레시피만 읽어주는 친구 같습니다. 재료를 사고, 썰고, 불에 올리는 건 결국 내가 해야 합니다.

클로드코드는 "옆에서 직접 일하는" AI입니다. 이 친구는 내 프로젝트 폴더를 통째로 열어봅니다. 어떤 파일들이 있는지, 그 안에 뭐가 적혀 있는지를 스스로 읽습니다. 그리고 무엇을 어떤 순서로 고칠지 **계획을 세운 다음**, 파일을 직접 만들고 고칩니다. 새 파일을 저장하는 것도, 기존 파일을 수정하는 것도, 심지어 프로그램을 실행해서 잘 돌아가는지 확인하는 것도 그가 합니다. 레시피만 읽어주는 친구가 아니라, **주방에 같이 들어와서 실제로 요리하는 파트너**인 셈입니다.

이 차이를 한 문장으로 요약하면 이렇습니다.

> **챗봇 = 대화하는 AI · 클로드코드 = 직접 일하는 AI 파트너.**

전문가들은 이 클로드코드의 작동 방식을 **"읽고 → 계획하고 → 실행하는 루프(read → plan → do)"**라고 부릅니다. 어려운 말 같지만, 우리가 일 잘하는 사람에게 기대하는 바로 그 순서입니다. 무턱대고 손부터 대는 게 아니라, ① 먼저 상황을 파악하고(읽기) ② 어떻게 할지 그림을 그리고(계획) ③ 실제로 손을 대는(실행) 순서. 그리고 실행하다가 막히면 다시 읽고, 다시 계획하고, 다시 실행합니다. 이 루프가 계속 돌아갑니다.

특히 반가운 점 하나. **한국어로 시켜도 됩니다.** "냉장고 사진을 올리면 레시피를 추천해 주는 앱을 만들어 줘"처럼, 그냥 우리말로 원하는 걸 말하면 됩니다. 코딩 언어를 배울 필요가 없습니다. 실제로 이 책의 여러 참고 강의에서도 강사들이 처음부터 끝까지 한국어 문장으로만 지시하며 앱을 완성하는 모습을 보여줍니다.



⚙️ **더 깊이** — 클로드코드는 컴퓨터의 터미널(검은 화면에 글자를 치는 창)에서 실행되는 프로그램입니다. 실행하면 지금 열려 있는 폴더를 기준으로 작동하기 때문에, 만들고 싶은 프로젝트 폴더 안에서 켜는 것이 기본입니다. 파일을 읽고 쓰는 것뿐 아니라, 필요하면 스스로 명령어를 실행해 프로그램을 돌려보고 결과까지 확인합니다. 즉 "글자로 답만 주는" 게 아니라 "내 컴퓨터에서 실제로 손을 움직이는" 것이 챗봇과의 결정적 차이입니다.

그래서 뭐가 대단한가

복붙이 사라지면 무엇이 바뀔까요. 단순히 손이 편해지는 정도가 아닙니다. **머릿속 생각과 실제 결과물 사이의 거리가 확 줄어듭니다.** "이런 기능이 있으면 좋겠는데"라는 생각을, 코드를 배우거나 개발자를 구하지 않고도, 그 자리에서 말로 시켜 눈앞에 띄워볼 수 있게 됩니다. 사주시바가 하루 만에 만들어질 수 있었던 것도, 사람이 코드를 한 줄 한 줄 타이핑한 게 아니라 클로드코드가 읽고-계획하고-실행하는 루프를 빠르게 반복했기 때문입니다. 이건 뒤 챕터에서 자세히 보여드리겠습니다.

클로드코드를 더 강하게 만드는 네 가지 부품

클로드코드는 그냥 쓰기만 해도 훌륭한 일꾼입니다. 하지만 진짜 힘은 여기에 몇 가지 장치를 얹었을 때 나옵니다. 이 책 전체에서 계속 등장할 네 가지 개념 — **스킬, 커맨드, MCP, 서브에이전트** — 을 지금 한자리에 모아 소개하겠습니다. 지금은 "아, 대충 이런 거구나" 정도만 잡으셔도 충분합니다. 실제로 어떻게 쓰였는지는 뒤 챕터에서 사주시바를 해부하며 하나씩 보게 됩니다.

🧩 **스킬 (Skill)**

AI에게 주는 **업무 매뉴얼**. 한 번 써두면 누가 해도 같은 품질. 평소엔 이름만, 필요할 때만 펼쳐 가볍다.

⌨️ **커맨드 (Command)**

나만의 **단축 지시어**. 반복 작업을 단어 하나로. 예: "배포" → 테스트·저장·배포·보고.

🔌 **MCP**

AI에 끼우는 **특별한 부품**(레고 특수 블록·연장통). 도구가 늘면 할 수 있는 일이 는다.

🏃 **서브에이전트**

심부름 시키기. 별도 작업공간을 가진 도우미에게 맡기고 결과만 받는다.



스킬(Skill) — AI에게 주는 '업무 매뉴얼'

회사에 신입사원이 들어왔다고 생각해 보세요. 일을 시킬 때마다 "이건 이렇게, 저건 저렇게" 매번 말로 설명하면, 시키는 사람도 지치고 결과물도 그때그때 들쭉날쭉합니다. 그래서 회사는 **업무 매뉴얼**을 만듭니다. 한 번 잘 써두면, 누가 하든 일정한 품질이 나오니까요.

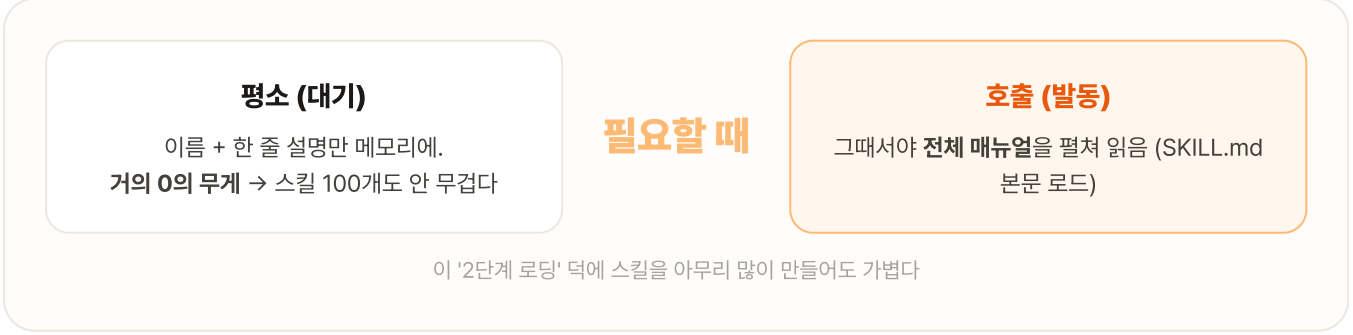
스킬이 바로 이 업무 매뉴얼입니다. 클로드코드에게 "썸네일 만들 땐 이 색을 쓰고, 이 폰트를 쓰고, 이런 순서로 검수해"라는 절차를 한 번 문서로 적어두면, 다음부터는 "썸네일 만들어"라는 말 한마디에 매번 같은 품질로 만들어 냅니다. 오늘은 잘 나오고 내일은 이상하게 나오는 일이 없어집니다. 참고 강의의 한 강사는 이걸 이렇게 표현했습니다.

> "프롬프트가 한 번의 지시라면, 스킬은 재사용 가능한 업무 시스템입니다."

스킬에는 아주 영리한 장치가 하나 숨어 있습니다. 바로 **2단계 로딩**입니다. 이게 스킬의 진짜 핵심이에요.

- **1단계 (평소)**: 클로드코드는 이 스킬의 **이름과 한 줄 설명만** 기억하고 있습니다. "썸네일-생성: 유튜브 썸네일을 브랜드 규칙대로 만드는 매뉴얼" 딱 이 정도. 아주 가볍습니다.
- **2단계 (필요할 때)**: 실제로 썸네일을 만들 상황이 오면, **그제야** 매뉴얼 전체를 펼쳐 읽습니다. 다 쓰고 나면 다시 덮어둡니다.

이게 왜 중요할까요. 매뉴얼을 100개 만들어 두더라도, 평소엔 100개의 "제목"만 책상 위에 올려두는 거라 전혀 부담이 없기 때문입니다. 실제로 이 책의 근간(뼈대)인 사주시바 프로젝트에는 스킬이 열아홉 개나 들어 있는데도, 클로드코드가 무거워지지 않습니다. 필요한 매뉴얼만 그때그때 펼쳐 보기 때문입니다.



평소 (대기)

이름 + 한 줄 설명만 메모리에.
거의 0의 무게 → 스킬 100개도 안 무겁다

필요할 때

호출 (발동)

그때서야 **전체 매뉴얼**을 펼쳐 읽음 (SKILL.md 본문 로드)

이 '2단계 로딩' 덕에 스킬을 아무리 많이 만들어도 가볍다

⚙️ **더 깊이** — 스킬은 폴더 하나로 관리됩니다. 폴더 안의 `SKILL.md` 라는 파일이 그 스킬의 본체(매뉴얼)이고, 맨 위에 이름과 설명이 적혀 있습니다. 평소 클로드코드가 상주시키는 건 이 이름+설명뿐이라 스킬 하나당 대략 수백 바이트(짧은 설명 한두 문장) 수준입니다. 매뉴얼 본문이 아무리 길어도(수만 자여도) 호출 시점에만 읽어들이므로, 스킬을 20개, 30개 만들어도 평소 부담이 거의 없습니다. 사주시바의 `.claude/skills/` 폴더에는 `shorts-produce`, `brand-verify`, `script-gen`, `voice-gen`, `render-shorts` 등 실제로 열아홉 개의 스킬이 들어 있습니다.

한 가지 덧붙이면, 스킬은 **한 번 만들고 끝이 아닙니다.** 쓰다 보면 "이 부분은 이렇게 바꾸는 게 낫겠다" 싶은 게 생기고, 그때마다 매뉴얼을 조금씩 다듬습니다. 처음부터 완벽할 필요는 없고, 쓰면서 갈고닦아 가는 물건입니다.

커맨드(Command) — 나만의 '단축 지시어'

매뉴얼과 헷갈리기 쉬운 게 커맨드인데, 결이 조금 다릅니다. **커맨드는 나만의 단축 지시어입니다.** 여러 단계를 거쳐야 하는 반복 작업을, 한 단어로 부르는 거예요.

예를 들어 제가 하루 작업을 마칠 때마다 ① 테스트를 돌리고 ② 변경사항을 저장하고 ③ 서버에 올리고 ④ 결과를 보고받는 네 단계를 매번 거친다고 해봅시다. 이걸 커맨드로 묶어두면, `/done` 이라고 한 단어만 치면 이 네 단계가 알아서 착착 진행됩니다. 실제로 사주시바에는 `/done` 이라는 커맨드가 있어서, 이 한 단어로 "모노레포 전체 커밋·푸시 + 웹 배포 + 문서 정합성 검증"이 한 번에 돌아갑니다. 매번 여러 명령을 순서대로 외워서 칠 필요가 없어지는 거죠.

말로 시키는 것(예: "이제 저장하고 배포해 줘")과 뭐가 다른가 싶으실 텐데, 커맨드의 장점은 **안정성**입니다. 말은 그때그때 표현이 달라지지만, 커맨드는 미리 정해둔 절차를 정확히 같은 방식으로 실행합니다. 자주 하는 일일수록 단어 하나로 딱 부르는 게 실수도 적고 빠릅니다.

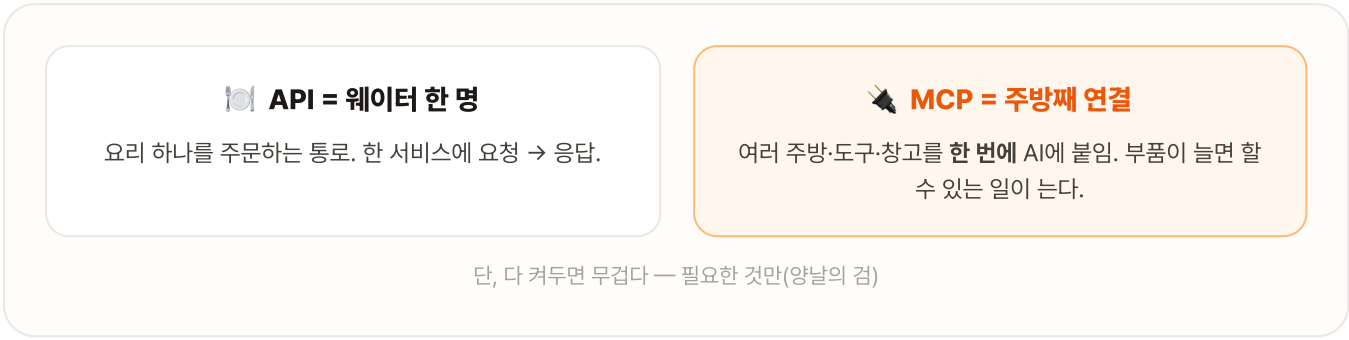
⚙️ **더 깊이** — 커맨드는 프로젝트 폴더 안의 `.claude/commands/` 폴더에 `이름.md` 형태로 저장됩니다. 사주시바에는 `done`, `ingest`, `query`, `lint`, `wiki-refresh` 다섯 개의 커맨드가 있습니다. 스킬과의 차이를 거칠게 나누면, **스킬은 "어떤 일을 어떻게 하는지에 대한 매뉴얼(품질 보장)"**, **커맨드는 "여러 단계를 한 단어로 묶은 지름길(반복 자동화)"**에 가깝습니다. 실무에서는 커맨드가 내부적으로 여러 스킬을 불러 쓰기도 합니다.

MCP — AI에게 끼우는 '특별한 부품'

클로드코드는 기본적으로 파일을 읽고 쓰는 일에 능합니다. 그런데 "노션에 정리해서 저장해 줘"라거나 "이 웹페이지를 실제 브라우저로 열어서 버튼이 잘 눌리는지 테스트해 줘" 같은, **바깥 세상의 도구를 다뤄야 하는 일**은 어떻게 할까요. 이때 끼우는 게 **MCP**입니다.

MCP는 레고에 끼우는 **특별한 부품**이라고 생각하시면 쉽습니다. 아니면 목수의 **도구상자**를 떠올려도 좋습니다. 망치도 있고, 톱도 있고, 드릴도 있죠. 이런 도구를 많이 쥐여줄수록 클로드코드가 할 수 있는 일이 늘어납니다. 노션 부품을 끼우면 노션에 글을 쓸 수 있고, 브라우저 부품을 끼우면 웹페이지를 직접 열어 테스트할 수 있고, 데이터베이스 부품을 끼우면 클라우드에 데이터를 저장할 수 있게 됩니다.

그런데 여기서 자주 나오는 질문. "그거 그냥 API 아니에요?" 참고 강의는 이 둘의 차이를 **식당 웨이터** 비유로 설명합니다. **API**는 식당에서 특정 메뉴 하나를 주문할 때 부르는 웨이터 같은 존재입니다. "이 서비스의 이 기능 하나만 빌려 쓰겠다"는 거죠. 반면 **MCP**는 여러 웨이터(여러 API·도구)를 표준화된 방식으로 한꺼번에 부릴 수 있게 해주는 규격입니다. 하나만 쓰는 게 아니라, 여러 외부 도구를 클로드코드가 자연스럽게 갈아 끼워 쓸 수 있게 해주는 통로인 셈입니다.



다만 중요한 주의사항이 하나 있습니다. **MCP는 양날의 검입니다.** 도구를 많이 끼울수록 할 수 있는 일은 늘지만, 부품을 끼워둔 것만으로도 클로드코드가 계속 신경을 써야 해서 **몸이 무거워집니다.** 안 쓰는 도구까지 잔뜩 켜두면 오히려 굼떠지죠. 그래서 실무에서는 "정말 필요한 부품만 골라 끼우고, 안 쓰는 건 빼둔다"가 기본 원칙입니다. 이 "가볍게 유지하기"의 감각은 이 책 내내 반복해서 등장하는 주제입니다.

> ⚙️ **더 깊이** — MCP는 Model Context Protocol의 줄임말로, AI가 외부 데이터·도구·서비스와 연결될 때 쓰는 표준 통신 규약입니다(Anthropic이 제안). 참고 강의에서는 Notion(문서 저장), Playwright(브라우저 자동 테스트), Supabase(클라우드 DB), GitHub(코드 보관), Context7(최신 문서 조회) 같은 MCP를 실제로 끼워 씁니다. 부품 하나를 끼울 때마다 클로드코드가 그 도구의 사용법을 항상 기억해야 하므로, 켜둔 MCP가 많을수록 평소 부담(컨텍스트 점유)이 커집니다. 앞서 본 스킬이 "쓸 때만 펼치는" 방식이라 훨씬 가벼운 것과 대비됩니다.

서브에이전트 — '심부름 시키기'

마지막 부품은 **서브에이전트**입니다. 이건 아주 직관적입니다. 어떤 일을 할 때 우리는 늘 두 가지를 저울질합니다. **"내가 직접 할까, 아니면 남한테 심부름을 시킬까?"**

간단한 일은 내가 직접 하는 게 빠릅니다. 그런데 시간이 오래 걸리는 일을 내가 붙잡고 있으면, 그동안 다른 걸 못 합니다. 그럴 땐 심부름을 시키고 결과만 받는 게 낫죠. **서브에이전트가 바로 이 심부름꾼입니다.** 메인 클로드코드가 "파일 100개를 다 뒤져서 이런 코드가 있는지 찾아와"처럼 오래 걸리는 조사를 통째로 서브에이전트에게 맡기면, 그 지저분한 조사 과정은 심부름꾼의 머릿속에서만 벌어지고, 메인에게는 **깔끔하게 요약된 결과만** 돌아옵니다.

여기에 두 가지 큰 장점이 있습니다.

- **병렬 처리** — 심부름꾼 여럿에게 서로 다른 일을 동시에 시킬 수 있습니다. 백엔드 담당, 프론트엔드 담당, 테스트 담당을 한꺼번에 돌리면 전체 시간이 확 줄어듭니다.
- **컨텍스트 보호** — 메인 클로드코드의 "작업 책상"이 잡다한 조사 내용으로 어질러지지 않습니다. 심부름꾼이 자기 책상에서 다 정리하고 요약만 가져오니까요.

판단 기준은 간단합니다. **"이 일이 독립적인가?"** 서로 얽히지 않고 따로 굴러갈 수 있는 일이면 심부름(서브에이전트)이 좋습니다. 반대로 순서대로 착착 이어가야 하거나 모든 맥락을 공유해야 하는 일은, 굳이 나누지 말고 메인에서 직접 하는 게 낫습니다. 사주시바의 콘텐츠 공장이 여러 조각을 나눠 처리하는 방식(CH4에서 자세히)도 이 "직접 할까 vs 시킬까"의 감각 위에 서 있습니다.



원래부터 들어있는 기능 — /goal 과 /loop

스킬·커맨드가 "내가 만들어 넣는" 장치라면, 클로드코드에는 **처음부터 기본으로 들어있는** 기능도 있습니다. 그중 이 책에서 자주 나올 두 가지가 `/goal`과 `/loop`입니다. 여기서 딱 짚고 넘어갈 게 하나 있어요. **이 둘은 제가 만드는 게 아니라, 클로드코드에 원래 있는 공식 기능입니다.** 뒤에서 "goal로 시작해서 loop로 돌렸다"는 문장이 나와도, '무슨 특별한 걸 만들었나?' 하고 오해하지 않으셨으면 합니다.

- `/goal` **— 목표 선언(킥오프)**: "우리가 지금부터 이걸 만들 거야"라고 목표를 명확히 세워두고 작업을 시작하는 기능입니다. 프로젝트의 출발선을 긋는 셈이죠.
- `/loop` **— 끝까지 반복 실행**: 정해둔 일을 다 끝날 때까지 지치지 않고 반복하는 기능입니다. 예를 들어 할 일 목록 54개가 있으면, 하나씩 처리하며 "안 함 → 하는 중 → 완료"로 표시를 바꿔가면서, 전부 완료될 때까지 스스로 돌립니다.

사주시바를 하루 만에 완주한 비결의 한 축이 바로 이겁니다. 목표를 `/goal`로 딱 세워두고, 해야 할 일들을 `/loop`로 끝까지 밀어붙인 것. 이 실제 사이클은 CH3에서 화면 보듯 풀어드리겠습니다.

잊지 말아야 할 세 가지 원칙

부품 소개는 끝났습니다. 마지막으로, 참고 강의들이 입을 모아 강조하는 **세 가지 원칙**을 짧게 심어두겠습니다. 이 세 문장은 이 책 전체를 관통하는 뼈대라서, 지금 외워두시면 뒤가 훨씬 수월합니다.

첫째, "컨텍스트가 왕이다." 클로드코드에게는 한 번에 담아둘 수 있는 기억의 크기(작업 책상의 넓이)가 정해져 있습니다. 이 책상이 잡다한 것으로 꽉 차버리면, AI가 앞에서 무슨 얘길 했는지 까먹고 엉뚱한 결과를 냅니다. 그래서 **깨끗하고 필요한 것만 올라온 책상**이, 이것저것 잔뜩 쌓인 책상보다 훨씬 낫습니다. 앞에서 본 스킬의 2단계 로딩도, MCP를 필요한 것만 끼우라는 조언도, 서브에이전트로 조사를 떼어내는 것도, 전부 이 "책상을 깨끗이 지키자"는 원칙에서 나온 겁니다.

🎯 컨텍스트가 왕	1️⃣ 한 세션 = 한 기능	🩹 에러는 그대로
AI가 한 번에 기억할 공간은 유한. 채울수록 흐릿해진다.	한 대화엔 한 가지 일만. 끝나면 새로 (신선한 컨텍스트).	막히면 에러 메시지를 그대로 붙여넣기 → 스스로 고침

둘째, "한 세션 = 한 기능." 한 번에 하나의 기능에만 집중해서 만들고, 끝나면 책상을 한번 정리하고 다음 기능으로 넘어가는 습관입니다. 이것저것 동시에 벌이면 책상이 금세 어질러지니, "이번 판은 이 기능 하나"라고 범위를 좁히는 거죠. 이 원칙은 CH3에서 사주시바를 한 걸음씩(Phase별로) 쌓아 올릴 때 그대로 나타납니다.

셋째, "에러는 그대로 붙여넣으면 스스로 고친다." 초보자가 가장 겁내는 순간이 "에러 메시지"가 뜰 때인데, 클로드코드에겐 이게 오히려 반가운 먹잇감입니다. 그 시뻘건 에러 메시지를 해석하려 애쓰지 말고, **통째로 복사해서 그대로 붙여넣으세요.** 클로드코드는 그 메시지를 읽고 원인을 짚어 스스로 고칩니다. 참고 강의에서도 강사들이 에러를 감추지 않고 일부러 보여주며, 그대로 붙여넣어 해결하는 장면을 몇 번이고 반복합니다. **에러는 실패가 아니라, 다음 수리 지시서**인 셈입니다.

그래서 뭐가 대단한가 — 그리고 다음 이야기

여기까지 오셨으면, 이제 낯선 단어의 벽은 웬만큼 넘으셨습니다. 정리하면 이렇습니다. 클로드코드는 대화만 하는 챗봇이 아니라 **내 폴더를 읽고-계획하고-실행하는 파트너**이고, 여기에 **스킬(매뉴얼)·커맨드(단축 지시어)·MCP(특별 부품)·서브에이전트(심부름)**를 얹으면 그 파트너가 점점 더 유능한 팀이 됩니다. `/goal`과 `/loop` 같은 기본 기능으로 목표를 세우고 끝까지 밀어붙이고, "컨텍스트가 왕", "한 세션 한 기능", "에러는 그대로 붙여넣기" 세 원칙으로 그 팀을 잘 부립니다.

이게 왜 대단하냐면 — 이 개념들 덕분에 **비전공자 한 사람이 "팀 하나 몫"의 일을 해낼 수 있게** 되기 때문입니다. 실제로 이 책의 근간(뼈대)인 사주시바는, 이 단어들을 조합해 웹서비스 하나와 콘텐츠 공장 하나를 만들어냈습니다. 지금은 이 단어들이 조금 추상적으로 느껴질 수 있지만, 뒤 챕터에서 실제 파일과 실제 화면을 보며 하나씩 짚어드리면 "아, 그래서 그 단어였구나" 하고 또렷해질 겁니다.

그런데 이 부품들을 낱개로 아는 것과, 이걸 **매일 알아서 굴러가는 시스템으로 엮는 것**은 또 다른 이야기입니다. 다음 챕터에서는 그 차이 — **한 번 만드는 '레시피(워크플로우)'와, 매일 요리하고 태운 접시를 잡아내는 '주방(하네스)'의 차이**부터 짚어보겠습니다.

CH2 — 워크플로우 vs 하네스: 레시피와 주방

앞 장에서 클로드코드가 "read → plan → do 루프를 도는 일꾼"이라는 걸 봤습니다. 그런데 일꾼 한 명에게 "영상 한 편 만들어줘"라고 시키는 것과, 그 일꾼이 **매일 아침 알아서 세 편씩 뽑아 다섯 플랫폼에 올려놓는 것**은 완전히 다른 이야기입니다. 앞엣것은 한 번의 부탁이고, 뒤엣것은 **공장**입니다.

이 책 전체를 관통하는 두 단어가 여기서 갈립니다. **워크플로우(workflow)** 와 **하네스(harness)**. 이 두 단어만 몸에 익히면, 사주시바가 어떻게 "잠자는 동안에도 콘텐츠가 발행되는" 상태에 도달했는지가 통째로 이해됩니다. 어렵게 들리지만, 사실은 여러분이 이미 아는 두 가지입니다. **레시피**와 **주방**이죠.

한 줄로 먼저 못을 박고 시작하겠습니다.

> **워크플로우는 레시피, 하네스는 매일 요리하고 태운 접시를 잡아내고 같은 접시를 두 번 안 내는 주방입니다.**

워크플로우 = 레시피

레시피가 뭘까요. "재료를 이 순서로, 이렇게 다뤄서, 이걸 만든다"를 **사람 말로 적어둔 것**입니다. 김치찌개 레시피에는 "① 돼지고기를 볶는다 → ② 김치를 넣고 더 볶는다 → ③ 물을 붓고 끓인다 → ④ 두부를 올린다"처럼, **무엇을(what) 어떤 순서로(order)** 하는지가 적혀 있죠. 코드 한 줄 없이도 사람이 읽고 따라 할 수 있습니다.

워크플로우도 똑같습니다. **"무슨 작업을 어떤 순서로 하고, 각 단계가 만들어낸 산출물이 다음 단계에게 상태를 넘겨준다"** 를 사람 말로 설계해 둔 흐름입니다. 여기서 두 가지가 핵심입니다. 하나는 **순서(사람 말로 쓴 흐름)**, 다른 하나는 **아티팩트 릴레이** — 즉 앞 단계가 남긴 파일을 뒷 단계가 이어받는 **바통 넘기기**입니다.

사주시바의 숏폼 영상 한 편을 만드는 워크플로우는 이렇게 생겼습니다. 여덟 단계의 레시피죠.

기획 → 대본 → 화면 → 목소리 → 검수 → 렌더 → 캡션 → 발행

한글로만 읽어도 무슨 요리인지 감이 오시죠. 각 단계를 사람 말로 풀면 이렇습니다.

- **① 기획** — "내일 띠별 운세로 영상 하나 만들자"라는 아이디어를 받아서, 어떤 형식으로 만들지·훅(첫 1초에 붙잡는 한 마디)·마무리 멘트를 정합니다.
- **② 대본** — 화면에 깔릴 나레이션과 자막 문장을, 각 장면이 몇 초짜리인지까지 씁니다.
- **③ 화면** — 열두 띠의 점수·한 줄 운세 같은 실제 데이터를 영상 템플릿에 꽂아 넣습니다.
- **④ 목소리** — 대본을 성우 목소리(TTS)로 읽혀서 음성 파일을 만듭니다.
- **⑤ 검수** — 완성 직전 화면을 세 장 캡처해서, 강아지 마스코트가 글자를 가리진 않는지·브랜드 색이 맞는지 눈으로 확인합니다. **여기가 사람이 딱 한 번 손대는 지점입니다.**
- **⑥ 렌더** — 실제 세로 영상(mp4) 파일로 구워냅니다.
- **⑦ 캡션** — 유튜브·틱톡·인스타 등 플랫폼마다 올릴 제목·설명·해시태그를 뽑습니다.
- **⑧ 발행** — 각 플랫폼에 실제로 업로드합니다.

여기까지가 "레시피"입니다. 놀라운 점은, 이 여덟 단계가 서로 이야기하는 방식이에요. 각 단계는 옆 단계에게 말로 소리쳐 넘기지 않습니다. `spec.json` **이라는 종이 한 장에 자기가 한 일을 적어서 다음 사람에게 넘깁니다.**

이게 바로 **아티팩트 릴레이**입니다. 기획 단계가 `spec.json`에 "주제·형식·훅"을 적어 넣으면, 대본 단계가 그 종이를 읽고 자기 칸("captions")에 대본을 채워 넣습니다. 화면 단계는 다시 그 종이를 읽고 데이터를 꽂고, 캡션 단계는 또 그 종이를 읽고 제목·태그를 채웁니다. **하나의 종이가 단계를 거치며 점점 살이 붙어 완성되는 것**이죠. 요리로 치면 주문서 한 장이 조리대를 따라 이동하면서 "고기 볶음 완료", "간 맞춤 완료" 도장이 하나씩 찍히는 것과 같습니다.



여기서 실리콘밸리 개발자들이 즐겨 쓰는 프레임 하나가 딱 맞습니다. 그들은 AI 자동화를 **워크플로우 · 에이전트 · 툴 (WAT)** 세 층으로 나눠 보는데, 그중 첫 번째 원칙이 **"흐름은 사람 말로 설계한다"** 입니다. 위 여덟 단계 어디에도 어려운 코드는 없었죠? 김치찌개 레시피처럼, 비전공자도 읽고 이해할 수 있는 **말의 순서**입니다. 이게 워크플로우의 정체입니다. 레시피는 종이에 적힌 말일 뿐, 아직 아무 요리도 안 됐습니다. 불을 켜고 실제로 매일 요리하는 건 다음 이야기, **주방**의 몫입니다.

하네스 = 주방

레시피가 아무리 완벽해도, 종이만으로는 저녁이 차려지지 않습니다. 누군가 **매일 그 레시피대로 실제로 요리**해야 하고, 손님이 몰려도 순서가 안 꼬여야 하고, 중간에 전화 받느라 멈췄어도 **하던 데서 이어서** 해야 하고, 접시를 태웠으면 **그 요리만 다시** 하고, 같은 테이블에 똑같은 접시를 **두 번 내보내면 안 됩니다.** 이 모든 걸 해내는 물리적 공간과 장치가 **주방**입니다.

하네스가 바로 그 주방입니다. 레시피(워크플로우)를 매일 실제로 돌리고, 멈춰도 재개하고, 실패하면 그 단계만 다시 하고, 같은 걸 두 번 안 내보내는 **기계장치의 총합**이죠. 하네스라는 영어 단어 자체가 원래 "말(馬)에 채우는 마구(馬具)"를 뜻합니다. 힘 센 말이 아무 데로나 날뛰지 않게 **방향을 잡아주고 안전하게 부리는 장치**죠. AI라는 힘 센 말에게 채우는 안전장치 — 그게 하네스입니다.

사주시바의 주방은 네 개의 장치로 이루어져 있습니다.

장치 ① 오케스트레이터 — 매일 요리하는 총괄 요리사

`shorts-produce`라는 오케스트레이터가 총괄 요리사입니다. "이 기획으로 영상 뽑아"라고 한마디 던지면, 이 녀석이 레시피 여덟 단계를 **순서대로 직접 구동**합니다. 기획 스킬을 부르고, 그 결과를 받아 대본 스킬을 부르고, 화면·목소리·검수·렌더·캡션·발행을 차례로 지휘하죠. 요리사 한 명이 여덟 개 조리대를 오가며 주문서(`spec.json`)를 챙기는 그림입니다.

중요한 건, 이 요리사가 **거대한 만능 로봇 한 대가 아니라는 점**입니다. 각 단계는 `content-plan`, `script-gen`, `data-fill` 처럼 **작고 독립적인 조각 스킬 일곱 개**로 쪼개져 있고, 오케스트레이터는 그저 이들을 순서대로 부를 뿐입니다. 이건 WAT 프레임의 또 다른 원칙 — **"작은 도구 여러 개가 거대한 하나보다 안정적이다"** — 를 그대로 구현한 것입니다. 대본 뽑는 로직이 잘못돼도 대본 조각만 손보면 되지, 전체 공장을 뜯지 않아도 되니까요.

장치 ② 계약서 — 모든 조각이 참조하는 단일 규격서

조각 스킬 일곱 개가 제각각 놀면 주방은 아수라장이 됩니다. 그래서 사주시바 주방에는 **표준 규격서가 딱 한 장** 붙어 있습니다. `CONTRACT.md`, 말 그대로 **공유 계약서**죠. "주문서(`spec.json`)는 이렇게 생겨야 하고, 파일은 여기에 두고, 영상은 반드시 세로 1080×1920으로 굽고, 강아지 마스코트는 좌하단에 두되 절대 글자를 가리지 마라" 같은 규칙이 전부 여기 한 곳에 적혀 있습니다.

모든 조각은 **자기 마음대로 하지 않고 이 계약서를 먼저 읽습니다.** 계약서와 어긋나면? 계약서를 먼저 고칩니다. 이게 주방의 질서를 유지하는 **단일 진실(single source of truth)** 입니다. 요리사가 열 명이어도 "우리 집 김치찌개 규격"이 벽에 딱 한 장 붙어 있으면 맛이 흔들리지 않는 것과 같죠.

장치 ③ 재개 장치 — 멈춰도 하던 데서 이어서

레시피 여덟 단계 중 다섯 번째에서 컴퓨터가 꺼지거나 작업이 끊겼다고 합시다. 처음부터 다시? 아닙니다. 사주시바 주방은 각 단계가 끝날 때마다 **"여기까지 했음"을 `status.json` 이라는 진행표에 도장 찍어 둡니다.** 다시 켜면 오케스트레이터가 이 진행표를 읽고, **이미 끝난 단계는 건너뛰고 멈춘 지점부터 이어서** 요리를 재개합니다.

주방으로 치면, 요리사가 잠깐 자리를 비웠다 와도 조리대에 붙은 진행 카드를 보고 "아, 간까지 맞췄으니 이제 두부만 올리면 되네" 하고 이어가는 것과 같습니다. 처음부터 다시 볶지 않죠.

장치 ④ 재시도 장치 — 태운 접시를 잡아내고, 그 요리만 다시

검수 단계(`brand-verify`)에서 "강아지가 3위 글자를 가렸다"는 불합격이 나오면? 영상 전체를 처음부터 다시 만들지 않습니다. 하네스는 **불합격 사유를 보고 원인이 된 단계 하나만 콕 집어 다시 돌립니다.** 자막이 문제면 화면 조각만, 대본이 너무 길면 대본 조각만 다시 하죠. 이걸 **백트래킹(되짚어 고치기)** 이라 부르고, **최대 두 번까지** 스스로 시도합니다. 두 번 고쳐도 안 되면 그때서야 사람에게 "이건 제가 못 고치겠어요" 하고 손을 듭니다.

이게 WAT 프레임의 세 번째 원칙 — **"막히면 스스로 고친다"** — 입니다. 태운 접시가 나왔을 때 주방 전체를 멈추는 게 아니라, 그 접시만 잡아내서 다시 만드는 거죠.

장치 ⑤ 스케줄러 — 매일 정해진 시각에 자동으로, 같은 접시는 두 번 안 냄

여기까지가 "한 편을 제대로 만드는" 주방이었다면, 진짜 공장은 **매일 저절로 돌아가는 것**입니다. `shorts-scheduler` 라는 스케줄러가 그 역할을 합니다. 정해진 요일·시각이 되면 알아서 데이터를 새로 뽑아 영상을 굽고 발행까지 끝냅니다. 사람이 버튼을 안 눌러도, **자는 동안에도** 말이죠.

그런데 자동으로 여러 번 돌리다 보면 무서운 일이 생깁니다 — 같은 영상을 두 번 올리는 것. 그래서 스케줄러에는 원장(ledger) 이라는 장부가 있습니다. "이번 주 띠별 운세는 이미

발행함"이라고 장부에 적어두고, 다음에 또 돌아도 장부를 먼저 확인해 이미 낸 접시면 건너뜁니다. 그 결과가 계약서에 박제된 한 문장입니다 — "여러 번 실행해도 중복 발행 0." 손님 테이블에 똑같은 접시가 두 번 나가는 사고를 원천 봉쇄한 것이죠.

왜 굳이 주방까지 필요한가 — "AI의 판단"과 "실제 실행"을 분리하다

여기서 근본 질문. 클로드코드 같은 똑똑한 AI에게 그냥 "매일 영상 올려줘" 하면 안 되나요? 왜 계약서·원장·스케줄러 같은 딱딱한 장치를 굳이 두를까요?

AI를 다루는 방법론은 지난 몇 년간 계단을 밟아 올라왔습니다. 처음엔 **프롬프트 잘 쓰기**(질문을 잘하기), 다음엔 **컨텍스트 잘 채우기**(필요한 자료를 잘 물려주기), 그다음이 **바이브 코딩**(AI에게 통째로 맡기기)이었죠. 그런데 바이브 코딩엔 명확한 한계가 있었습니다. AI가 점점 자율적으로 움직일수록, **내가 의도하지 않은 방향으로 튕겨 나가거나 위험한 일을 저지르는 순간**이 생깁니다. 힘 센 말이 마구 없이 날뛰는 거죠.

그래서 등장한 게 **하네스 엔지니어링**입니다. 핵심 통찰은 이겁니다.

> **AI의 "판단"과 실제 "실행"을 분리하는 것이 안정성의 열쇠다.**

무슨 뜻일까요. "이번 영상 훅을 뭘로 할까", "자막을 어떻게 줄일까" 같은 **판단·창작**은 AI에게 맡깁니다(레시피를 짜는 머리). 하지만 "같은 영상을 두 번 올리지 마라", "영상은 반드시 세로 1080×1920이어야 한다", "검수 통과 전엔 발행 금지" 같은 **반드시 지켜야 할 실행 규칙**은 AI의 그때그때 기분에 맡기지 않고 **코드와 장치로 못 박아 강제**합니다. AI는 "그렇게 하는 게 좋겠다"고 유도될 뿐이지 100% 지킨다는 보장이 없지만, 원장·계약서·스케줄러 같은 **프로그래밍 로직은 예외 없이 강제**되니까요.

사주시바 주방을 다시 보면 정확히 이 분리가 보입니다. **창작이 필요한 부분**(기획·대본·화면 구성)은 조각 스킬 안에서 AI가 판단하고, **틀리면 안 되는 부분**(중복 방지·규격 검증·검수 게이트·매일 정시 발행)은 원장·계약서·스케줄러가 기계적으로 강제합니다. 힘 센 말에게 방향키(하네스)를 채운 것이죠.

📋 워크플로우 = 레시피

무엇을·어떤 순서로, 어떤 산출물이 상태를 이어받나

기획 → 대본 → 화면 → 목소리 → 검수 → 렌더 → 발행

↳ 사이를 잇는 파일: spec.json

🍳 하네스 = 주방

그 레시피를 매일 돌리고·재개·재시도·중복방지

오케스트레이터 공유 계약 CONTRACT

스케줄러 거버넌스

레시피는 **무엇을**, 주방은 **매일 실제로 돌리고 태운 접시를 잡아내는** 장치

그래서 뭐가 대단한가

이 두 단어의 구분이 왜 이 책의 뼈대일까요.

대부분의 사람은 워크플로우(레시피)까지만 만들고 멈춥니다. "이렇게 이렇게 하면 영상 한 편이 나온다"는 순서를 알아내는 것만으로도 대단하죠. 하지만 그건 **한 번 요리하는 것**입니다. 오늘 내가 앉아서 버튼을 눌러야만 결과물이 나옵니다. 손을 떼는 순간 공장은 멈춥니다.

사주시바가 남다른 지점은 **레시피에서 멈추지 않고 주방까지 지었다**는 것입니다. 계약서로 질서를 잡고, 진행표로 재개를 보장하고, 백트래킹으로 실패를 스스로 복구하고, 원장 달린 스케줄러로 매일 자동 발행하되 중복은 0으로 막았습니다. 그 결과가 **"사람이 딱 한 번(검수) 손대고, 나머지는 자는 동안에도 돌아가는 콘텐츠 공장"** 입니다.

레시피는 지식이고, 주방은 자산입니다. 레시피는 머릿속에 있지만, 주방은 내가 잠든 사이에도 돈을 법니다. 이 책이 진짜 보여주고 싶은 건 바로 **"어떻게 레시피를 주방으로 승격시켰는가"** 입니다.

그리고 좋은 소식 하나. 하네스는 처음부터 완벽하게 설계하는 게 아닙니다. **반복되는 작업을 커맨드로 → 커맨드를 스킬로 → 스킬 여럿을 워크플로우로 → 꼭 지킬 규칙을 강제 장치로**, 이렇게 **작은 것부터 하나씩 쌓아 올리면** 어느새 나만의 주방이 완성됩니다. 사주시바도 그렇게 지어졌습니다.

다음 챕터로

이제 두 단어가 손에 잡히셨을 겁니다. 다음 CH3에서는 이 이야기의 **첫 번째 산출물**, 즉 "사주 웹서비스를 하루 만에 만든 법"을 봅니다 — 레시피를 설계하고(plan-first), 킥오프 한 장으로 시작하고(goal), 54개 항목을 100% 채워 완주한(loop) 개발 하네스 이야기죠. 그리고 지금 배운 두 단어가 실제로 **어디서 살아 움직이는지**는 두 곳에서 다시 만납니다. **CH4(콘텐츠 자동화)** 에서는 방금 본 여덟 단계 레시피와 `spec.json` 릴레이가 어떻게 구현됐는지를, **CH6(무인화)** 에서는 원장 달린 스케줄러가 어떻게 "자면서 발행하는 3중 하네스"로 완성됐는지를 해부합니다. 레시피와 주방, 이 두 단어를 계속 붙들고 따라오세요.

CH3 — 사주 웹서비스는 어떻게 만들었나 (goal/loop 개발법)

로그인이 되고, 사주 계산이 진짜로 맞고, 실제 카드로 990원이 결제되고, 환불까지 되는 웹서비스. 보통 이런 걸 만들려면 몇 주, 몇 달을 잡습니다. 그런데 사주시바의 첫 버전은 **2026년 7월 2일 하루** 안에 여기까지 갔습니다. 3개의 기능(오늘의 무료운세·궁합·연도별 운세)과 그 뒤를 받치는 만세력 엔진, AI 해석, 구글/이메일 로그인, 그리고 **실제 라이브 카드 결제**까지요.

이게 "천재가 손 빠르게 쳐서" 된 일이 아닙니다. 오히려 반대예요. 빠를 수 있었던 이유는 **처음부터 순서를 지켰기 때문**입니다. 이번 챕터는 그 순서를 통째로 해부합니다. 클로드코드를 실제로 이렇게 굴리면 어떤 그림이 나오는지, 여러분이 그대로 따라 할 수 있게요.

핵심 흐름은 여섯 단계입니다. **경쟁사 역설계 → 설계문서 먼저(plan-first) → goal(킥오프 프롬프트 한 장) → loop(체크리스트 100%까지) → Phase 게이트(검증 통과해야 다음) → 하루 완주.** 그리고 이 서비스의 진짜 심장인 **"계산과 AI를 물리적으로 갈라놓은 구조"**까지, 하나씩 보겠습니다.



1. 시작은 "빈 상상"이 아니라 "경쟁사 해부"

많은 사람이 새 서비스를 만들 때 백지에서 시작합니다. "사주 앱을 만들자, 뭐부터 넣지?" 하고 머릿속으로 기능을 상상하죠. 이게 가장 흔한 실패의 씨앗입니다. 상상으로 만든 기능은 대개 너무 많고, 정작 돈이 되는 지점은 빠져 있거든요.

사주시바는 정반대로 시작했습니다. 이미 잘 돌아가는 경쟁 서비스인 **레퍼런스 사이트**를 통째로 뜯어봤어요. 클로드코드가 MCP(브라우저를 직접 조작하는 도구)로 그 사이트를 **깊이 크롤링**하면서, 어떤 페이지가 있고, 어디서 돈을 받고, 무료로 뭘 풀고, 결과물이 어떻게 생겼는지를 전부 기록했습니다. 그 결과를 두 개의 문서로 **박제**했어요. 하나는 사이트 해부 보고서, 하나는 위키 페이지(`wiki/competitors/레퍼런스-사이트.md`)입니다.

⚙️ **더 깊이** — 경쟁사 해부는 `docs/plans/레퍼런스-사이트-분석-보고서.md` (UX·페이월·화폐·결과물 실측)와 위키 페이지 `wiki/competitors/레퍼런스-사이트.md` (사이트맵·2종 페이월 패턴·충전팩 990~9,900원)로 분리 저장됐습니다. 실제로 레퍼런스 사이트의 유료 화폐를 결제해 궁합 결과물까지 역설계했고, 결과 페이지의 버튼 클래스가 `shadcn/ui` 기본 컴포넌트임까지 확인해 "1:1 이식이 쉽다"는 판단 근거로 삼았습니다.

이 해부에서 딱 두 가지 리스트가 나왔습니다. **"만들 것 3개"** 와 **"피할 실패 3개"**.

만들 것 3개 — 레퍼런스 사이트의 여러 유료 기능 중에서, "수익화가 빠르고 구현이 간단한" 순서로 3개만 골랐습니다.

- **오늘의 무료운세** — 로그인도 필요 없는 미끼. 사람을 데려오고 붙잡는 역할.
- **궁합** — 감정 몰입이 가장 커서, 첫 결제를 이끌어낼 후보.
- **연도별 운세** — 지각가치가 높아 업셀(더 비싼 것 권하기) 후보.

그리고 사주 콘텐츠가 제일 많은 대표 유료 사주 상품, 날짜 로직이 복잡한 "택일", 채팅·통화 인프라가 필요한 "1:1 상담"은 **일부러 제외**했습니다. 전부 "빠르고 간단"에 역행하니까요.

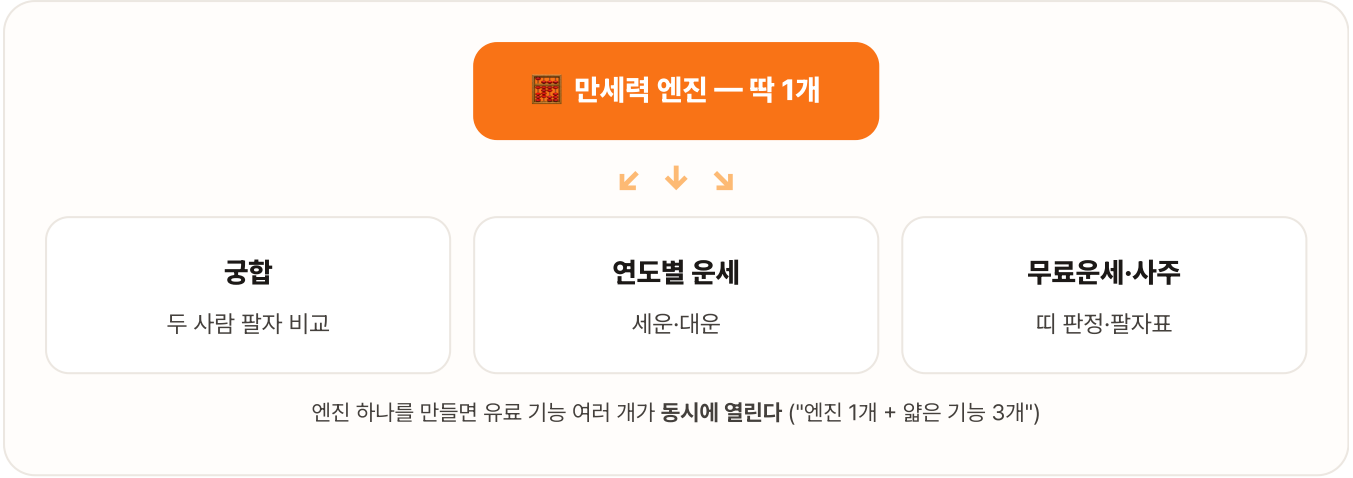
피할 실패 3개 — 경쟁사가 잘못한 지점을 그대로 뒤집었습니다.

- 레퍼런스 사이트는 궁합·사주 결과를 **0% 보여주고** 바로 "결제하세요"를 띄웁니다(하드 게이트). → 우리는 개인화된 미리보기 + 블러 처리된 점수를 보여주는 **소프트 티저**로.
- 레퍼런스 사이트는 **공유 기능을 결제 뒤에 잠가** 놓았습니다. 그래서 무료 바이럴이 죽었어요. → 우리는 공유를 **결제 앞으로** 빼서 무료로 열었습니다.
- 레퍼런스 사이트는 콜드 트래픽이 가장 먼저 만나는 대표 상품(990원 사주)이 하필 하드 게이트였습니다(설계 불일치). → 우리는 첫 관문을 가장 마찰 없는 무료운세로.

여기서 나온 결정적 통찰이 하나 있습니다. **유료 기능 3개가 전부 같은 부품 하나를 공유한다**는 것. 바로 **만세력 엔진**(생년월일시 → 사주팔자·대운을 계산하는 엔진)입니다. 궁합도, 연도별도, 나중에 붙일 대운·사주도 전부 이 엔진 위에 얹힙니다. 그래서 결론이 이렇게 정리됐어요.

> **"엔진 1개를 제대로 만들면, 그 위에 얇은 기능 3개를 얹기만 하면 된다."**

이게 하루 완주의 진짜 비결입니다. 만들 게 4개(엔진+기능3)로 보이지만, 실제 어려운 건 엔진 딱 하나뿐이고 나머지는 그 위에 얹는 껍데기라는 걸 **시작 전에 알아냈다**는 거죠.



> ⚠️ 여기서 궁합을 "첫 결제 트리거", 연도별을 "업셀"로 배치한 것, 그리고 뒤에 나올 "첫 결제 0원 프로모"는 모두 **설계 의도(가설)** 입니다. 실제로 그 순서대로 사람들이 결제하는지, 전환율이 얼마인지는 이 시점에 검증된 게 아니에요. "이런 효과를 노렸다"까지가 사실이고, 그 효과가 났다고 말할 수 있는 단계는 아직 아닙니다.

그래서 뭐가 대단한가. 첫 삽을 뜨기도 전에, "무엇을 만들지 / 무엇을 만들지 않을지 / 어디가 병목인지"가 근거와 함께 확정됐다는 점입니다. 상상이 아니라 실측이 스펙을 만들었어요. 이게 없으면 나중에 "이 기능 넣을까 말까"로 하루를 다 씁니다.

---

2. 코드 0줄 전에, 설계문서부터 (plan-first)

경쟁사 해부가 끝나도 곧바로 코드를 치지 않았습니다. 대신 코드가 정말로 0줄인 상태에서 **설계문서 5종** — 경쟁사 해부, 이론 리서치, "무엇을·왜"를 담은 설계 허브, 엔진 구현 스펙, 결제 연동 설계 — 을 먼저 썼고, 문서끼리 "무엇이 무엇의 근거인지" 관계와 읽는 순서까지 묶어 뒀습니다.

왜 굳이 문서를 먼저 쓰나? 클로드코드 같은 AI 도구는 "무엇을 만들지"가 흐릿하면 흐릿한 대로 그럴듯하게 만들어 버립니다. 그리고 그 흐릿함은 나중에 "이게 아닌데?"로 돌아와요. 설계문서는 **AI에게 줄 정답지를 미리 만들어두는 일**입니다. 특히 사주처럼 계산이 복잡한 도메인에서는, 원칙("계산과 해석을 분리한다", "절기·진태양시는 타협 없다")을 문서에 박아두지 않으면 AI가 대충 지어냅니다.

그래서 뭐가 대단한가. 하루 만에 완주할 수 있었던 두 번째 비결이 여기입니다. 빌드 중에 "어? 이거 어떻게 하기로 했지?" 하고 멈추는 순간이 거의 없었어요. 답이 이미 문서에 있으니까요.

---

3. goal — 세션에 붙여넣는 "킥오프 프롬프트 한 장"

설계문서가 다 준비되면, 이제 **실제 빌드를 지시**할 차례입니다. 여기서 클로드코드의 `/goal` 개념이 등장합니다.

`goal` 은 거창한 게 아닙니다. **"이번 세션에서 무엇을, 어떤 순서로, 어떤 규칙을 지키며 만들지"를 한 장으로 압축한 프롬프트**예요. 사주시바에서는 이 한 장이 킥오프 문서 안에 그대로 들어 있어서, 새 세션에 붙여넣는 순간 AI가 뭘 해야 하는지·무엇을 읽어야 하는지·무엇을 넘지 말아야 하는지를 전부 알게 됩니다. 그리고 이 목표는 세션 내내 유지됩니다 — 옆길로 새다가도 항상 이 목표로 돌아옵니다. 특히 가드레일 중 **"미결/모호한 건 임의로 정하지 말고 물어볼 것"** 한 줄이 결정적이었는데, 이 이야기는 잠시 뒤 다시 나옵니다.

그래서 뭐가 대단한가. `goal`은 AI에게 주는 **북극성**입니다. 대화가 길어지고 복잡해져도 "우리가 원래 뭘 하려고 했지?"를 잃지 않게 하는 닻이에요. 사람도 그렇지만 AI는 특히 이 닻이 없으면 표류합니다.

---

4. loop — 체크리스트 54개를 100%까지 몰아붙이기

목표(goal)가 "어디로 갈지"라면, `loop`는 "다 갈 때까지 멈추지 않는 것"입니다.

사주시바의 빌드는 **54개의 원자 항목(더 못 쪼갤 만큼 작은 할 일)** 으로 된 체크리스트로 굴러갔습니다. 각 항목은 "아직 안 함 → 하는 중 → 완료"의 세 상태를 오가는데, '완료' 도장의 조건이 엄격합니다 — 그냥 "코드 짰음"이 아니라 **"작동을 실제로 확인했음"** 이어야 하죠. 그리고 `loop`의 핵심은 이겁니다. **54개가 전부 완료될 때까지 계속 돈다.** 중간에 "대충 이 정도면 됐지"로 멈추지 않아요. 실제로 이 체크리스트의 마지막 줄은 이렇게 끝납니다.

> **"체크리스트 54/54 `[x]`. 미완료 0."**



그래서 뭐가 대단한가. `loop`는 "80%에서 지쳐 멈추는" 흔한 실패를 구조적으로 막습니다. 사람은 90% 지점에서 "거의 다 됐네"라며 손을 놓기 쉬운데, 남은 10%가 사실 서비스의 생사를 가르는 부분(결제 보안·에러 처리·접근성)인 경우가 많거든요. 체크리스트가 남은 `[ ]`를 계속 보여주는 한, 끝은 100%뿐입니다.

---

5. Phase 게이트 — 통과 못 하면 다음은 없다

가장 어려운 부품인 만세력 엔진은 그냥 한 번에 짜지 않았습니다. **달력 변환 → 팔자 배정 → 명리 파생 → 조립·API → AI 해석 골대, 이렇게 5개의 Phase로 쪼개고, 각 Phase 끝에 "검증 통과"를 다음 단계의 입장권으로** 걸었어요.

이 게이트의 심장은 **골든 테스트**입니다. 사주 계산은 답이 정해져 있어요. 1990년 5월 15일 오전 8시 30분생 남자의 사주 팔자는 **경오 / 신사 / 경진 / 경진**입니다. 이건 협상 대상이 아니에요. 그래서 엔진 구현 문서에는 이런 원칙이 대문짝만하게 박혀 있습니다.

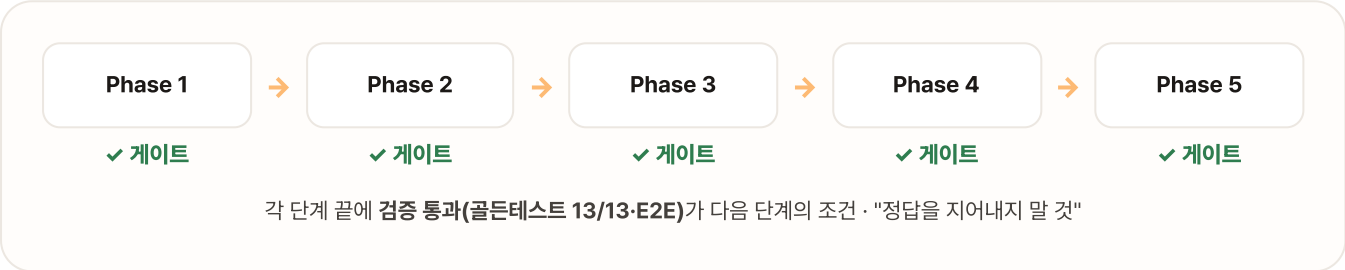
> **"정답 만세력을 임의로 지어내지 말 것."**

즉, AI가 "이 정도면 맞겠지" 하고 정답을 창작하는 걸 금지했습니다. 대신 검증된 만세력 라이브러리(`lunar-javascript`)를 정답원으로 삼고, 손으로 확인한 참조 명식 3개를 한글로 하드코딩해 문서화했어요. 그리고 **틀리기 쉬운 경계(엣지)**

케이스를 집중적으로 걸었습니다. 최종적으로 **13개 테스트가 13개 다 통과(13/13)** 해야 엔진이 완성된 걸로 쳤습니다.

그 엣지 케이스들이 이 엔진의 진짜 실력입니다.

- **입춘 절기 경계** — 1990년 입춘은 2월 4일 10시 14분. 그 전(오전 9시)에 태어나면 연주가 전년(기사), 후(오전 11시)면 당년(경오)로 **갈립니다**. 달력 날짜가 아니라 절기 절입 시각으로 해가 바뀌거든요.
- **야자시(밤 11시~자정)** — 23시 30분생과 다음날 0시 30분생은 일주가 다릅니다(경진 vs 신사). 이 미묘한 경계를 정확히 처리해요.
- **음력 윤달** — 음력 2020년 윤4월 15일 → 양력 2020년 6월 6일로 정확히 변환.
- **대운 방향** — 양남(양의 해에 태어난 남자)은 순행, 음남은 역행. 방향이 반대로 흐릅니다.
- **시 모름** — 태어난 시각을 모르는 사람은 시주를 생략하고 나머지만 정상 계산.



그래서 뭐가 대단한가. 사주 서비스에서 계산이 틀리면 그 순간 신뢰가 무너집니다. Phase 게이트는 "틀린 채로 다음으로 넘어가는 것"을 구조적으로 불가능하게 만들었어요. 그리고 이 방식은 사주에만 쓰이는 게 아닙니다. 어떤 복잡한 기능이든 "쪼개고, 각 조각의 정답을 미리 정하고, 통과해야 다음"이라는 틀은 그대로 이식됩니다.

---

6. 임의로 정하지 않는다 — 모호하면 멈추고 질문한다

빠르게 만들다 보면 유혹이 생깁니다. "이건 애매한데, 그냥 이렇게 하자." 사주시바는 이걸 **명시적으로 금지**했습니다. 킥오프 가드레일에 "미결·모호한 건 임의로 정하지 말고 물어볼 것"이 박혀 있고, 설계문서에는 아예 **"임의 결정 금지" 목록(Open Decisions)** 이 따로 있었어요.

대표적인 예가 **자시(子時) 처리**입니다. 밤 11시 이후 태어난 사람의 일주(그날의 기둥)를 그날로 볼지 다음날로 볼지는 사주 유파마다 다릅니다(야자시 vs 조자시). 이건 "AI가 알아서 하나 고르면" 되는 문제가 아니에요. 서비스의 계산 결과 전체가 달라지니까요.

그래서 AI는 이 지점에서 **멈추고 사용자에게 물었습니다.** 그리고 사용자가 "야자시로 간다"고 확정해준 뒤에야 진행했어요. 실제로 이 결정은 체크리스트에 `D1 자시: 사용자 확인 → 야자시 유지 확정`으로 기록됐습니다.

> ⚙️ **더 깊이** — 원래 엔진 설계문서의 기본안은 "조자시(23시=다음날 일주)"였는데, 실제 만세력 라이브러리는 자정에 일주가 바뀌는 방식(야자시)이었습니다. 이 불일치를 AI가 임의로 덮지 않고, 사용자에게 확인받아 **라이브러리 기본값(야자시)을 채택**했어요. 비슷하게, 타입 정의에서도 "태어난 시각을 모를 수 있다"는 걸 반영해 `hour: number`를 `hour: number | null`로 확장하는 작은 결정 하나까지 근거를 주석으로 남겼습니다.

그래서 뭐가 대단한가. "빠르게"와 "제대로"는 보통 상충한다고들 하는데, 이 프로젝트는 그 사이에 선을 하나 그었습니다. **뒤집기 쉬운 사소한 결정은 AI가 알아서, 서비스의 근간을 흔드는 결정은 반드시 사람에게.** 이 선 덕분에 빠르면서도 나중에 "아, 그때 그걸 물어봤어야 했는데"가 없었습니다.

---

7. ★ 이 서비스의 심장: 계산과 AI를 물리적으로 갈라놓기

이제 이 챕터에서 가장 중요한 부분입니다. 사주시바가 기술적으로 특별한 이유는 딱 하나로 요약됩니다.

> **결정론 계산과 AI 서술을, 서로 다른 모듈로 물리적으로 분리했다.**

무슨 말이냐면요. 사주 서비스에는 성격이 완전히 다른 두 종류의 일이 섞여 있습니다.

① 계산 — 답이 정해진 일. 생년월일시를 넣으면 사주팔자, 오행 분포, 십성, 대운이 나옵니다. 이건 1+1=2처럼 정답이 있어요. 같은 입력이면 항상 같은 출력이어야 합니다. 이걸 `src/lib/saju/` 폴더에 몰아넣었어요. 이 안의 코드는 **순수하고(랜덤·현재시각·네트워크 의존 0)**, **동기적이고(기다림 없이 즉시 계산)**, **서버 전용(server-only)** 입니다. 공개 진입점은 함수 하나, `computeSajuChart(생년월일시) → 사주명식`이에요.

② 서술 — 답이 없는 일. "이 사주를 사람 말로 어떻게 풀어줄까?" 이건 정답이 없습니다. 감성적이고 창의적이어야 하죠. 이걸 `src/lib/interpret/` 폴더에 따로 뒀습니다. 여기서 AI(구글 Gemini)가 일해요.

그리고 이 둘 사이에 **철칙**이 하나 있습니다.

> **"AI는 계산을 하지 않는다. 이미 계산된 값을 받아서, 해석하고 서술만 한다."**

즉, AI에게 "이 사람 사주 계산해서 궁합 봐줘"라고 시키지 않습니다. 그렇게 하면 AI가 사주팔자를 **지어내거든요**(이걸 환각이라고 합니다). 대신 이렇게 합니다. 먼저 결정론 엔진이 두 사람의 정확한 사주팔자·오행·신살을 계산해서, 그 값들을 **정리된 요약문**으로 AI에게 건넵니다. AI는 그 숫자와 용어를 **받아서 이야기로 풀 뿐**, 새로 계산하거나 수치를 창작하지 않아요.



⚙️ **더 깊이** — AI에게 건네는 요약문(`chartDigest`)은 이렇게 생겼습니다. `일간(나): 경 / 사주팔자 — 년:경오 월:신사 일:경진 시:경진 / 오행분포: 목1 화2 토1 금3 수1 / 신강/신약: strong / 대운 시작칸: 9세 임오(식신) / 세운... / 신살...` . AI 프롬프트에는 "너는 25년 경력의 명리학자 '도사 시바'다. 위 계산 결과를 바탕으로 해석하라. **팔자 재계산·수치 창작 금지**"가 명시돼 있어요. 모델은 `gemini-2.5-flash` , 창의성을 위해 temperature 0.9로 두되, 출력은 반드시 정해진 JSON 스키마(제목·점수·섹션·조언)로만 받게 강제합니다.

이 분리 구조가 낳는 이점이 세 가지 더 있습니다.

첫째, 캐시로 돈을 아낍니다. AI를 부를 때마다 비용이 듭니다. 그런데 같은 사주팔자·같은 날짜·같은 해석 종류라면 결과도 같겠죠? 그래서 이 세 가지를 **열쇠(키)** 로 만들어 Supabase의 `interpretations` 테이블에 저장해둡니다. 다음에 같은 요청이 오면 AI를 부르지 않고 저장된 걸 그냥 꺼내줘요. AI는 **저장고에 없을 때(캐시 miss)만** 호출됩니다.

둘째, AI가 죽어도 서비스는 산다. AI 서버가 먹통이거나 키가 없으면 어떻게 될까요? 보통 서비스는 그냥 에러가 납니다. 사주시바는 다릅니다. AI 호출이 실패하면 `rule-text.ts` 라는 **결정론 fallback**이 대신 작동해요. 오행 상생상극과 합충을 계산해서 "그럴듯한 해석문"을 규칙 기반으로 만들어냅니다. 깊이는 얕지만, **항상 뭔가는 나옵니다.** 화면이 비는 일이 없어요.

셋째, 계산은 재사용됩니다. 결정론 엔진이 순수·독립적이다 보니, 웹 화면뿐 아니라 **숏폼 영상 자동 생성 스케줄러**도 이 엔진을 그대로 가져다 씁니다. 매주 만세력 데이터를 뽑아 영상 콘텐츠를 만들 때 같은 계산 코드를 재활용하는 거죠(이건 다음 챕터들에서 자세히).

⚙️ **더 깊이** — 만세력 라이브러리(`lunar-javascript`)는 `src/lib/saju/calendar/` 안의 어댑터 뒤에 격리돼 있습니다. 바깥 코드는 우리가 정의한 도메인 타입만 보고, 라이브러리가 뭘 반환하든 신경 쓰지 않아요. 나중에 더 좋은 라이브러리가 나오면 어댑터 내부만 갈아끼우면 됩니다. 이걸 설계 원칙 P3("lib 스왑 가능")로 문서에 박아뒀습니다.

그래서 뭐가 대단한가. AI 서비스를 만들 때 가장 흔한 함정이 "AI에게 다 맡기는 것"입니다. 그러면 환각이 나고, 비용이 폭발하고, AI가 죽으면 서비스도 죽어요. 사주시바는 **"정확해야 하는 건 코드가, 감성적이어야 하는 건 AI가"** 라는 선을 그어서 이 셋을 한 번에 해결했습니다. 이 구조는 사주뿐 아니라 여러분이 만들 거의 모든 AI 서비스에 그대로 옮겨 붙일 수 있는 뼈대예요.

---

8. 실전에서 걸린 함정 두 개 (그리고 어떻게 잡았나)

이렇게 잘 짜여도 실전에서는 함정이 나옵니다. 사주시바 빌드 중 잡아낸 두 개를 소개할게요. 완벽해서 대단한 게 아니라, **잡아냈기 때문에** 대단한 겁니다.

함정 1 — 캐시가 남의 이름을 노출할 뻔했다. 앞에서 "같은 사주팔자면 캐시를 공유해 비용을 아낀다"고 했죠. 그런데 여기 함정이 있었어요. 해석문 안에는 사람 **이름**이 들어갑니다("서연님은..."). 만약 사주팔자만으로 캐시 열쇠를 만들면, 우연히

같은 팔자를 가진 다른 사람이 조회했을 때 **먼저 조회한 사람의 이름이 그대로 나오는** 사고가 납니다. 이걸 발견하고, 캐시 열쇠에 **이름과 옵션까지 포함**하도록 고쳤어요. 출력에 영향을 주는 모든 입력을 열쇠에 넣어야 한다는 원칙이죠.

함정 2 — 결제 없이 무료 크레딧을 적립할 수 있었다. 결제가 완료되면 크레딧을 적립해주는 `finalize_payment` 라는 데이터베이스 함수가 있습니다. 보안 감사 중에, 이 함수가 **누구나 직접 호출할 수 있는(PUBLIC)** 상태라는 걸 발견했어요. 즉, 결제를 하지 않은 익명 사용자가 이 함수를 직접 불러서 **공짜로 크레딧을 쌓을 수 있는 구멍**이 있었던 겁니다. 이걸 **서버 전용(service_role)** 으로 잠갔습니다.

> ⚙️ **더 깊이** — 수정 후 검증까지 실제로 했습니다. 익명 사용자가 `finalize_payment` 를 호출하면 `42501 permission denied` (권한 없음) 에러가 나는 걸 확인했어요. 같은 감사에서 개인정보 유출 위험(이름이 담긴 해석·공유카드를 클라이언트가 직접 읽는 경로)도 함께 잠가서, 덤프를 시도하면 빈 배열 `[]` 이 나오게 만들었습니다. 결제 보안의 다른 원칙들도 지켰어요. 가격은 클라이언트를 믿지 않고 **서버 DB가 진실**, 콜백은 같은 결제가 두 번 적립되지 않게 **멱등 처리**, 결제 완료는 결제사에 **재확인(readBill)** 후에만 인정.

그래서 뭐가 대단한가. 이 두 함정은 겉으로는 멀쩡히 작동하는 서비스에 숨어 있었습니다. 눈에 보이는 기능이 다 돌아도, 그 아래 보이지 않는 곳에 신뢰와 돈이 새는 구멍이 있었던 거죠. `loop` 가 100%를 강제하고 보안 감사를 체크리스트에 넣었기 때문에 출시 전에 잡혔습니다.

---

9. 그래서, 하루 안에 무슨 일이 벌어졌나

이 모든 게 **2026년 7월 2일 하루** 안에 일어났습니다. 타임라인으로 보면 이렇습니다.

- **M0** — 프로젝트 뼈대 + Supabase + 로그인(구글·이메일).
- **M1** — 오늘의 무료운세 + 공유카드 무료 개방(엔진 없이).
- **M2** — 만세력 엔진(골든테스트 13/13) + 결제선생 연동 + 궁합(소프트 티저 + AI 해석).
- **M3** — 연도별 운세(엔진 재사용) + 가격 앵커링 + 첫 결제 0원 프로모.
- **최종** — 통합 검증 + 보안 수정 + 접근성·성능·에러처리·SEO + 배포.



그리고 결과는 상상이 아니라 **실제 돈으로 증명**됐습니다. 결제 팩은 990원 / 2,970원 / 4,950원 / 9,900원 4종. 샌드박스(테스트) 환경에서 실제 카드로 결제해 승인을 받고 크레딧이 적립되는 걸 확인했고, 더 나아가 **진짜 라이브 환경에서 실제**

카드로 990원을 결제 해 크레딧 적립 → 환불까지 전 과정을 돌려봤어요. 데이터베이스의 크레딧 회수 정합성까지 맞는 걸 확인하고서요.

> ⚙️ **더 깊이** — 라이브 실결제는 사용자(운영자 본인)의 명시적 승인을 받은 뒤에만 진행했습니다. 이 프로젝트는 "결제 정산 주체가 확실하지 않으면 실결제 금지"를 가드레일로 뒀고, 사용자 확인 후에야 라이브 키를 켰어요. 결제 한 건이 통과했다는 건 곧, 결제창 호출 → 결제사 콜백 → 서버 재확인 → 크레딧 적립 → 환불이라는 **결제 루프 전체가 실제로 작동한다**는 뜻입니다. 여기까지가 하루 안에 검증됐어요.

정리하면, 하루 완주를 가능하게 한 건 속도가 아니라 **순서**였습니다.

1. **경쟁사를 해부해** 상상이 아닌 실측으로 스펙을 정했고,
2. **설계문서를 먼저 써서** 빌드 중 헤매지 않았고,
3. **goal 한 장으로** AI에게 흔들리지 않는 방향을 줬고,
4. **loop로** 54개를 100%까지 몰아붙였고,
5. **Phase 게이트로** 틀린 채 넘어가는 걸 막았고,
6. **모호하면 멈춰 물었고,**
7. **계산과 AI를 갈라놓아** 정확함과 감성, 그리고 비용과 안정성을 동시에 잡았습니다.

이 일곱 가지는 사주에만 통하는 비법이 아닙니다. 어떤 웹서비스를 만들든 그대로 적용되는 **개발 플레이북**이에요.

그런데 사주시바 이야기는 여기서 끝이 아닙니다. 제품이 완성됐다고 사람이 저절로 오지는 않으니까요. 다음 챕터에서는 이 서비스에 사람을 데려오기 위해, **똑같은 클로드코드로 콘텐츠(숏폼 영상)를 공장처럼 찍어내는 자동화**를 어떻게 만들었는지 해부합니다. 제품을 만든 그 손이, 이번엔 마케팅 공장을 짓습니다.

CH4 — 콘텐츠 자동화 = 영상 공장

앞 챕터에서 우리는 사주 웹서비스라는 "제품"을 하루 만에 완주했습니다. 그런데 제품을 만드는 것과, 그 제품을 세상에 알리는 것은 전혀 다른 문제죠. 아무리 잘 만든 사이트도 사람들이 모르면 없는 것과 같습니다.

보통은 여기서 지칩니다. 숏폼 영상 하나를 만들어 본 적 있으신가요? 소재를 정하고, 대본을 쓰고, 자막을 얹고, 목소리를 입히고, 화면을 편집하고, 제목·설명·해시태그를 고민하고, 유튜브·틱톡·인스타·페이스북·쓰레드에 각각 올리고... 영상 한 편에 반나절이 훌쩍 갑니다. 매일 올려야 한다면? 사람 손으로는 도저히 감당이 안 됩니다.

그래서 사주시바에서는 이 과정을 **하나의 "영상 공장"으로 만들었습니다.** 한 줄짜리 아이디어("내일 띠별운세")를 넣으면, 컨베이어 벨트를 타고 흘러가듯 대본→성우 목소리→화면→발행까지 자동으로 이어집니다. 사람은 딱 한 곳, "이 영상 이대로 내보내도 될까요?" 하는 최종 확인 지점에서만 개입합니다.

이 챕터는 그 공장의 설계도를 하나씩 뜯어봅니다. 놀라운 건, 이 공장이 어떤 대단한 인공지능 마법으로 돌아가는 게 아니라 **아주 단순한 원칙 하나** 위에 세워졌다는 점입니다.

공장의 제1원칙 — "각 일꾼은 파일을 읽고, 파일을 남긴다"

이 공장을 관통하는 설계 철학은 딱 한 문장입니다.

> **각 스킬은 파일을 읽고, 파일을 남긴다.**

무슨 뜻일까요. 공장의 일꾼(스킬)들은 서로 말로 소통하지 않습니다. 앞 일꾼이 작업을 끝내면 그 결과를 **파일로 책상 위에 올려두고** 자리를 뜹니다. 다음 일꾼은 출근해서 그 책상 위 파일을 읽고, 자기 일을 하고, 다시 결과를 파일로 남깁니다. 사람이 릴레이 경주에서 바통을 넘기듯, 여기서는 **파일이 바통**입니다. 그래서 이 방식을 '아티팩트 릴레이'(artifact relay, 산출물 이어달리기)라고 부릅니다.

이 단순한 원칙이 왜 강력할까요? 세 가지 선물을 줍니다.

- **어느 단계서든 멈출 수 있다.** 3번 일꾼까지 하고 컴퓨터를 꺼도, 3번의 결과 파일은 책상 위에 그대로 남아 있습니다.
- **어느 단계서든 다시 시작할 수 있다.** 다음에 켜면 4번 일꾼이 책상 위 파일을 보고 이어서 일합니다. 처음부터 다시 할 필요가 없죠.
- **한 단계만 따로 다시 돌릴 수 있다.** 5번 결과가 마음에 안 들면, 그 원인이 된 일꾼만 불러서 다시 시키고 나머지는 건드리지 않습니다.

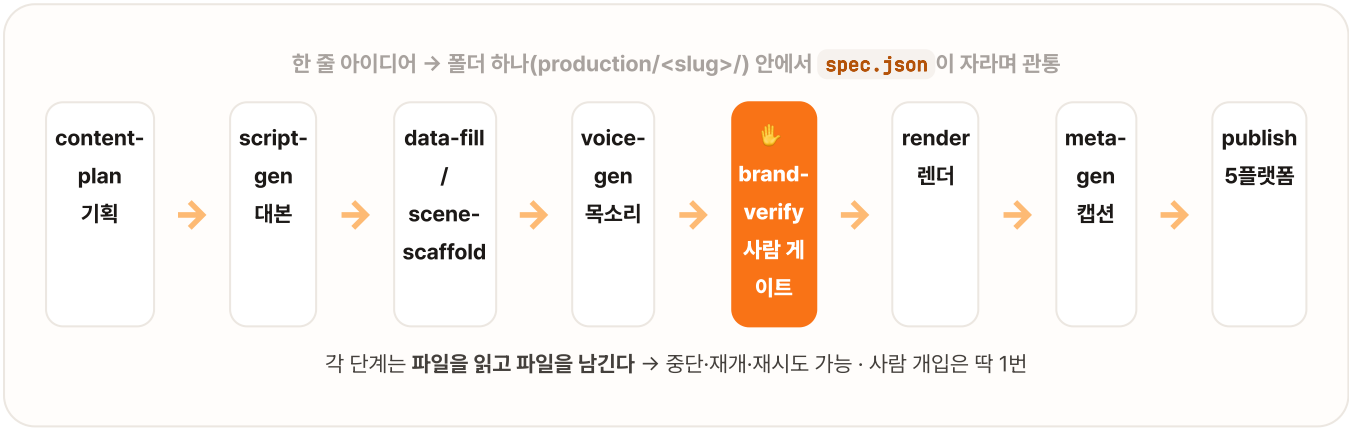
한 편의 영상은 폴더 하나로 관리됩니다. `production/2026-07-03-zodiac/` 같은 폴더 안에, 이 릴레이의 모든 바통(파일)이 차곡차곡 쌓입니다.

바통의 정체 — `spec.json`, 단계마다 자라나는 설계도

릴레이의 바통 중에서도 가장 중요한 것이 `spec.json`이라는 파일입니다. 이건 그냥 데이터 파일이 아니라, **단계를 거칠수록 점점 살이 붙는 설계도**입니다.

처음 1번 일꾼이 만들 때 `spec.json`은 뼈대만 있습니다. 주제가 뭔지, 어떤 형식인지, 어떤 데이터를 담을지 정도죠. 2번 일꾼이 대본을 써서 `captions` 항목을 채우고, 뒤로 갈수록 성우 정보, 발행 제목·해시태그가 덧붙습니다. 마지막에 이르면 하나의 완결된 명세서가 됩니다.

여기서 아주 영리한 규칙이 하나 있습니다. **각 단계는 자기가 맡은 칸만 채우고, 남의 칸은 절대 건드리지 않습니다.** 그리고 `spec.json`은 어느 순간에 열어봐도 **항상 말이 되는(유효한) 상태**여야 합니다. 그래서 중간에 멈춰도 파일이 깨지지 않고, 다음 일꾼이 안심하고 이어받을 수 있는 겁니다.



컨베이어 벨트 위의 8단계

이제 공장 라인을 따라 걸어봅시다. 오케스트레이터(현장 반장) 스킬 하나가 아래 8단계를 순서대로 지휘하고, 각 단계는 전담 조각 스킬 한 명이 처리합니다.

1. **content-plan (기획 → 명세서).** 사람이 던진 자유로운 한 줄("내일 띠별운세")을 받아 `spec.json`의 뼈대를 세웁니다. 이때 가장 중요한 판단을 내립니다 — 이 소재가 **기존 템플릿 재활용**으로 되는지, 아니면 **새 장면을 짜야** 하는지(뒤에서 자세히).
2. **script-gen (대본).** `spec`의 데이터·훅·마무리 멘트를 읽어 `captions[]`를 채웁니다. 여기서 만든 나레이션 문장이 곧 화면 자막이 됩니다 — 말과 글자가 따로 놀지 않죠.
3. **분기 — data-fill 또는 scene-scaffold.** 기존 템플릿이면 `data-fill`이 데이터·카피·대본만 갈아끼우고, 새 장면이면 `scene-scaffold`가 컴포지션을 새로 만듭니다.
4. **voice-gen (목소리).** Typecast 음성 합성으로 대본을 성우 목소리로 바꿔 **로컬 오디오 파일**로 저장합니다.
5. **brand-verify (검수) — ★유일한 사람 게이트.** 영상에서 스틸 3장(초반·중반·후반)을 뽑아 사람에게 보여주고 **승인을 기다립니다.** 통과 전까지는 렌더도 발행도 못 합니다.
6. **render-shorts (영상 완성).** Remotion으로 실제 mp4 영상을 뽑고, 파일이 규격에 맞는지 자동 검사합니다.
7. **meta-gen (발행 문안).** 제목·설명·해시태그·고정 댓글까지, 플랫폼별 발행 캡션 한 팩을 생성합니다.
8. **publish-shorts (발행).** 5개 플랫폼에 올립니다. 기본값은 비공개(private) — 공개는 명시적으로 지시할 때만. 여기서 한 가지 우아한 디테일. 대본의 각 문장이 몇 초짜리인지(`durationSec`)만 정하면, 영상 전체 길이가 그 합으로 **자**

동으로 결정됩니다. 문장을 하나 더 넣으면 영상이 그만큼 길어지고, 빼면 짧아집니다. 사람이 "이 영상 총 몇 초로 할까" 계산할 필요가 없어요.

사람은 딱 한 번만 개입한다 — brand-verify

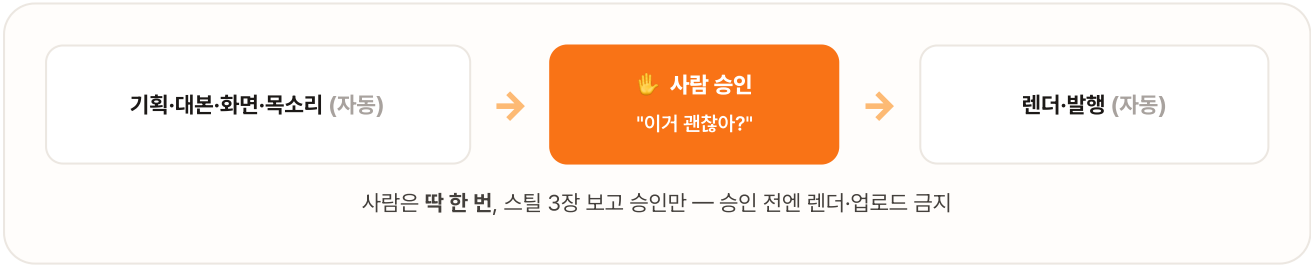
8단계 중 사람이 손을 대는 곳은 **오직 5번, 단 한 곳**입니다. 나머지 7단계는 전부 자동으로 흘러갑니다. 왜 하필 여기에 게이트를 두었을까요?

영상을 다 렌더하고 5개 플랫폼에 올린 다음에 "어? 마스코트가 1위 이름을 가리고 있네" 하고 발견하면, 이미 늦습니다. 지우고 다시 올려야 하죠. 그래서 **가장 비싼 작업(렌더·발행) 바로 앞에서** 사람이 최종 눈으로 확인하도록 게이트를 배치했습니다. 목소리까지는 자동으로 오지만, 실제 영상을 뽑기 직전에 스틸 3장으로 "이대로 갈까요?"를 묻는 거죠.

이 검수 단계의 성격이 재밌습니다. **brand-verify는 판단만 하고, 절대 파일을 고치지 않습니다.** 의사가 진단은 하되 수술은 담당 외과의에게 넘기듯, 이 스킬은 "무엇이 잘못됐고, 누구를 다시 불러야 하는지"만 리포트에 적습니다. 실제 수정은 원인이 된 조각 스킬이 맡습니다. 역할을 이렇게 딱 쪼개두면, 검수하는 눈과 고치는 손이 섞이지 않아 실수가 줄어듭니다.

검수 체크리스트는 브랜드 규칙을 지키는지 봅니다.

- 좌하단 마스코트(앉은 시바 강아지)가 **리스트·자막·제목을 가리지 않는가.**
- **로또/번호/당첨/회차** 같은 금지 단어가 한 글자도 없는가.
- 자막이 잘리지 않고 온전히 보이는가.
- 브랜드 색(흰 배경·골드·먹색·주사색)만 썼는가.
- 제목·데이터가 안전 영역 안에 있는가.



사람은 **딱 한 번**, 스틸 3장 보고 승인만 — 승인 전엔 렌더·업로드 금지

멈춰도, 틀려도 괜찮은 공장 — 재개와 백트래킹

좋은 공장은 실수를 안 하는 공장이 아니라, **실수해도 무너지지 않는 공장**입니다. 이 영상 공장에는 두 개의 안전장치가 있습니다.

첫째, 재개. 각 단계는 끝날 때마다 `status.json`의 완료 목록(`done[]`)에 자기 이름을 적습니다. 나중에 같은 영상을 다시 돌리면, 이미 끝난 단계는 "아 이건 이미 했네" 하고 건너뜁니다. 예를 들어 명세서가 있고 대본까지 채워졌으면 1·2단계는 스킵하고 3단계부터 시작하죠. 반나절 걸린 작업을 처음부터 다시 하는 참사가 없습니다.

둘째, 백트래킹(되짚어 고치기). 검수(brand-verify)에서 불합격이 나오면, 앞서 말한 `route` 표시를 따라 **문제를 일으킨 그 단계 딱 하나만** 다시 실행합니다. 자막이 너무 길어 잘렸다면 대본 담당(script-gen)을, 데이터에 로또 흔적이 남았다면

데이터 담당(data-fill)을 부르는 식입니다. 그리고 목소리·검수(4·5단계)를 다시 돌려 재확인합니다. 전체를 처음부터 다시 만드는 게 아니라, 고장 난 부품만 갈아 끼우는 거죠.

이 되짚기는 **최대 2번까지만** 자동으로 시도합니다. 2번을 고쳐도 여전히 불합격이면 기계가 무한 반복에 빠지지 않고 멈춰서, 스틸과 사유를 사람에게 그대로 보여줍니다. "여기까지 자동으로 해봤는데 안 됩니다, 봐주세요." 자동화가 폭주하지 않도록 안전 브레이크를 걸어둔 셈입니다.



이 공장이 만들 수 있는 것 vs 없는 것

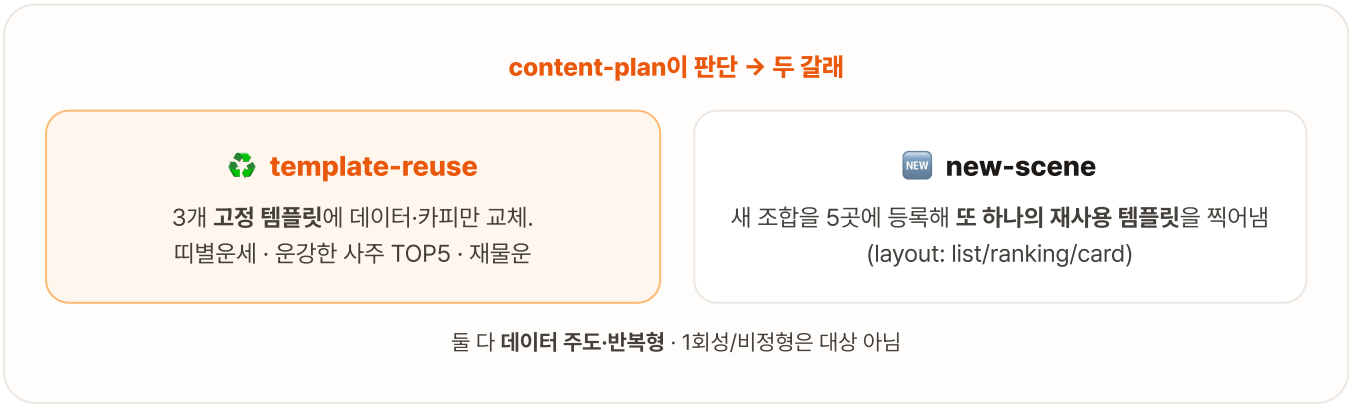
여기가 이 챕터에서 가장 중요한 대목입니다. **"영상을 자동으로 뽑는다"는 말에 속으면 안 됩니다.** 이 공장은 아무 영상이나 찍어내는 만능 기계가 아닙니다. **정해진 틀에 데이터만 갈아 끼우면 되는** 반복형 콘텐츠에만 작동합니다. 그리고 그게 바로 이 설계의 핵심 통찰이에요.

자동화의 갈림길(content-plan의 판단)은 두 갈래입니다.

(A) 템플릿 재활용(template-reuse). 이미 만들어 둔 **3개의 고정 템플릿** 중 하나에 데이터만 갈아 끼웁니다. 코드는 손도 안 대고, 콘텐츠·카피·대본만 교체하죠.

- **띠별 오늘의 운세**(ZodiacFortuneShorts) — 12띠 각각의 점수와 한 줄 운세
- **이번 주 운 강한 사주 TOP5**(WeeklyStrongSajuShorts) — 랭킹 리스트
- **사주팔자 재물·총운**(SajuPaljaShorts) — 재물운 카드

(B) 새 장면 만들기(new-scene). 위 셋 어디에도 안 맞는 새 조합이면, 새 컴포지션을 정해진 절차대로 등록해서 만듭니다. 그런데 여기 반전이 있습니다. new-scene은 그 영상 한 편만 위한 일회성 작업이 **아닙니다.** 이것도 결국 **"또 하나의 재사용 템플릿"을 찍어내는** 작업입니다. 한 번 새 장면을 만들어 두면, 그다음부터는 그것도 (A)처럼 데이터만 갈아 끼워 반복 생산할 수 있게 되죠. 공장에 새 생산 라인 하나를 증설하는 것과 같습니다.



그럼 **무엇이 자동화되고, 무엇이 안 될까요?**

- **된다 = 데이터 주도·반복형·정형 포맷.** 운세 리스트, 랭킹, 카드처럼 "스키마(빈칸이 정해진 표)를 채우고 렌더한다"로 환원되는 콘텐츠. 매주·매일 같은 틀에 새 데이터만 넣으면 되는 것들입니다.
- **안 된다 = 1회성·비정형 콘텐츠.** 이번 한 번만 쓸 특별한 기획, 정형화되지 않은 이야기. 이런 건 이 공장 라인에 태우지 않고 **별도의 발행 대기함(staging 큐)**에 수동으로 쟁여둡니다.
- **절대 안 된다 = 로또 프레임.** 로또·번호·당첨·회차라는 단어는 **모든 단계에서 금지**입니다. 데이터에도, 대본에도, 발행 제목·태그에도 한 글자도 들어가면 안 됩니다.
- **원리상 불가 = 결정론으로 못 뽑는 것.** 만세력 계산으로 딱 떨어지게 나오지 않는, 근거 없이 지어내야 하는 내용은 애초에 이 공장의 재료가 아닙니다(CH3에서 본 '계산과 서술의 분리' 철학이 여기까지 이어집니다).

이 경계선이 왜 중요할까요. **자동화가 잘 되는 콘텐츠의 조건을 정확히 알기 때문에**, 사주시바는 처음부터 "매주 반복 가능한 정형 포맷"으로 소재를 설계합니다. 만들기 어려운 걸 자동화하려 애쓰는 게 아니라, **자동화가 쉬운 형태로 콘텐츠 기획 자체를 정렬**한 거죠. 이게 순서가 거꾸로인 것 같지만, 실은 대량 생산의 정답입니다.

마지막 단계 — 5개 플랫폼에 한 번에

완성된 영상은 발행(publish-shorts) 단계에서 **5개 플랫폼**으로 나갑니다. 유튜브, 틱톡, 인스타그램, 페이스북, 쓰레드. 사람이 각 앱을 열어 다섯 번 업로드하는 대신, 명령 한 줄로 "무엇을 · 어디에" 올릴지만 정하면 됩니다.

기본 공개 설정은 **비공개(private)**입니다. 자동으로 만들었다고 곧바로 세상에 뿌리는 게 아니라, 일단 비공개로 올려두고 사람이 확인한 뒤 공개하는 안전한 기본값이죠. 공개하려면 명시적으로 지시해야 합니다. 물론 이렇게 대량으로 여러 채널에 organic(광고비 없이 자연 유입)으로 뿌리는 것이 **"낮은 비용으로 도달을 늘린다"**는 노림수인 것은 맞지만, 그 효과가 실제로 얼마나 나오는지는 아직 데이터를 쌓는 중입니다. 이 부분의 마케팅 가설과 측정은 다음 챕터에서 정직하게 다룹니다.

그래서 뭐가 대단한가

한 걸음 물러서서 보면, 이 영상 공장의 진짜 대단함은 화려한 기술이 아닙니다. **단순한 원칙 몇 개를 끝까지 지킨 것**이 대단합니다.

- **파일 릴레이** 하나로 "멈춤·재개·부분 재실행"이라는 세 가지 자유를 얻었습니다. 반나절 작업이 중간에 끊겨도 손실이 없습니다.
- **사람 게이트를 딱 한 곳**으로 압축했습니다. 자동화하되 눈은 가장 결정적인 순간에만 씁니다. 사람의 시간이 가장 비싼 자원이니까요.
- **자기수정 루프**(백트래킹 최대 2회)로 웬만한 실수는 기계가 스스로 고치고, 정말 안 되는 것만 사람에게 올립니다.
- **자동화의 경계를 명확히** 그었습니다. "데이터 채우고 렌더"로 환원되는 정형 콘텐츠만 공장에 태우고, 그렇게 되도록 콘텐츠 기획 자체를 설계했습니다.

결과적으로, 영상 한 편에 반나절 걸리던 일이 **"한 줄 아이디어 → 스틸 3장 확인 → 5플랫폼 발행"**으로 압축됩니다. 사람은 창의적인 소재를 던지고 마지막에 눈으로 한 번 확인할 뿐, 그 사이의 지루한 조립은 전부 공장이 합니다. 이것이 클로드코드

로 만든 콘텐츠 자동화의 실체입니다 — 마법이 아니라, **잘 쪼갠 일꾼들과 잘 넘긴 바통.**

이제 영상이 대량으로 생산돼 5개 플랫폼으로 흘러나가기 시작했습니다. 그렇다면 이 많은 콘텐츠는 무엇을 노리고, 어떻게 사람을 제품으로 데려오려 했을까요? 다음 챕터에서 그 **마케팅 퍼널의 원리와 노림수**를 — 확정된 성과가 아니라 '설계 의도'로서 — 정직하게 풀어봅니다.

CH5 — 마케팅 자동화, 원리와 노림수

앞 챕터(CH4)에서 우리는 "영상을 찍지 않고 코드로 렌더하는 숏폼 공장"을 봤습니다. 하루에 사람이 한 손가락도 안 대고 mp4 하나가 뚝 떨어지는 그 공장이요. 그런데 여기서 자연스럽게 이런 질문이 떠오릅니다.

"그래서 그 영상들, 왜 만드는 건데요?"

영상을 잘 만드는 것과, 그 영상으로 돈을 버는 것은 완전히 다른 이야기입니다. 아무리 예쁜 스코어카드를 하루 세 편씩 찍어내도 그게 사람을 데려오지 못하면, 데려온 사람이 지갑을 열지 않으면, 그냥 예쁜 쓰레기일 뿐이니까요.

이 챕터는 사주시바가 **"왜 이렇게 만들었나"** 를 다룹니다. 마케팅 파이프라인을 하나하나 뜯어보되, 각 장치가 어떤 **노림수(가설)** 를 품고 설계됐는지를 같이 읽습니다. 그리고 아주 중요한 고백을 하나 하겠습니다. 이 마케팅 기계는 아직 "성과를 낸 기계"가 아니라 **"데이터를 만들어서 검증하려고 돌리기 시작한 베팅"** 입니다. 이 점을 정직하게 밝히지 않으면 이 챕터 전체가 거짓말이 됩니다. 그래서 미리 못을 박아둘게요.

> ⚠️ **이 챕터를 읽는 원칙** — 여기 나오는 "효과"는 전부 **노렸다 / 가설이다** 로 읽으셔야 합니다. CAC(고객 한 명 데려오는 비용)·전환율·리텐션 같은 핵심 지표는 아직 신뢰할 만한 데이터로 확립되지 않았습니다. 뒤에서 다시 설명하지만, GA4 실데이터도 아직 안 쌓였습니다. "이렇게 하면 이런 효과가 날 것이다"라고 설계했을 뿐, "이렇게 했더니 이런 효과가 났다"가 아닙니다.

미션 자체가 마케팅 엔진이다

보통 우리는 "제품을 먼저 만들고, 그다음에 마케팅을 붙인다"고 생각합니다. 사주시바는 순서가 조금 다릅니다. 이 프로젝트의 뿌리에 있는 문장을 그대로 옮기면 이렇습니다.

> ⚙️ **더 깊이** — `saju_workspace/AGENTS.md` (프로젝트 헌법)의 Mission 항목: "경쟁 사주/운세 서비스를 분석하고, **인스타 릴스 자동화로 신규 트래픽을 만들며**, 방문자의 **첫 결제(최소 결제의지)를 낮은 마찰로 유발할 기능을 설계한다.**" 즉 마케팅(트래픽 생성)과 전환(첫 결제)이 프로젝트의 존재 이유 그 자체로 못박혀 있습니다.

정리하면 사주시바의 미션은 두 축입니다. **(1) 릴스 자동화로 신규 트래픽을 만든다. (2) 그 방문자가 최소한의 마찰로 첫 결제를 하게 만든다.** 제품(웹서비스)은 이 두 가지를 실현하기 위한 그릇이고요.

브랜드는 **도사 시바** 라는 강아지 캐릭터를 앞세운 `@sajushiba` 입니다. 그리고 이 브랜드는 한 곳이 아니라 **다섯 플랫폼** 에서 동시에 목소리를 냅니다. 유튜브·틱톡·인스타그램·페이스북·쓰레드. 왜 다섯 개나 되는지는 뒤에서 설명하겠습니다. 먼저 짚어둘 함정 하나.

⚙️ **더 깊이** — 이 책 곳곳에 등장하는 **레퍼런스 사이트**는 사주시바의 자산이 **아닙니다.** 벤치마킹 대상, 즉 뜯어보고 배우는 경쟁사입니다. AGENTS.md에 "레퍼런스 사이트는 우리 것이 아니라 벤치마킹 대상 경쟁사다 — 분석·복제스펙 참조 대상일 뿐, 우리 자산의 라이브 링크로 쓰지 않는다"라고 명시돼 있습니다. 이 챕터에서 "경쟁사는 이렇게 하던데 우리는 반대로 했다"는 이야기가 여러 번 나오는데, 그 경쟁사가 바로 이 레퍼런스 사이트입니다.

퍼널 3층 — 트래픽이 흘러가는 물길

마케팅을 처음 설계할 때 가장 먼저 그려야 하는 건 **퍼널(funnel)**, 즉 "낯선 사람이 어떤 물길을 따라 흘러 내려와서 결국 지갑을 여는가"입니다. 깔때기처럼 위는 넓고 아래로 갈수록 좁아진다고 해서 퍼널이라고 부릅니다. 사주시바의 퍼널은 3층으로 되어 있습니다.



[A] 맨 위 — 대량 콘텐츠 생산

퍼널의 맨 위 입구는 **콘텐츠**입니다. 낯선 사람이 스크롤하다가 손가락을 멈추게 만드는 그 짧은 영상이요. 사주시바는 이 영상을 **코드로 렌더링**합니다(CH4에서 자세히 다뤘죠). Remotion이라는 도구로 만든 숏폼인데, 화면 포맷이 고정돼 있습니다.

총점 + 세부 점수 + 럭키 아이템 + 반말톤 캡션 + 해시태그 — 이 다섯 요소가 한 장의 "릴스 스코어카드"로 조합됩니다. 예를 들면 "오늘 너의 총운은 78점, 애정운 90점, 금전운 62점, 오늘의 럭키 아이템은 파란 우산" 같은 식이죠. 반말톤인 이유는 도사 시바가 친근한 강아지 캐릭터라서, 그리고 짧은 영상에서 존댓말보다 반말이 더 훅이 세기 때문입니다.

⚙️ **더 깊이** — 이 포맷은 즉흥적으로 정한 게 아니라 **경쟁사가 검증한 포맷에 정렬**한 것입니다. 기획 문서(`wiki/features/첫결제-트리거-개선.md`)에 "레퍼런스 사이트 **오늘의 무료운세**(총점+5세부점수+럭키아이템+해시태그+반말톤)를 릴스 정적 스코어카드로 대량 생성"이라고 적혀 있습니다. 남이 이미 조회수로 검증해 놓은 포맷의 **껍데기(구조)** 만 빌려오고, 안에 채우는 **데이터는 사주시바 자체 만세력 엔진**으로 계산한다는 원칙입니다.

여기서 핵심 단어는 **"대량"** 입니다. 사람이 영상 편집기를 열어서 한 편씩 만들면 하루 한 편도 벅찹니다. 코드로 렌더하면 열 편, 백 편, 세그먼트별로 무한히 찍어낼 수 있습니다. 이 "대량"이 왜 마케팅의 핵심 무기인지는 뒤 '원리' 섹션에서 다시 짚겠습니다.

[B] 가운데 — 5플랫폼 organic 배포

영상 한 편을 렌더했으면, 이제 그걸 사람들 눈앞에 뿌려야 합니다. 여기서 사주시바는 아주 영리한 구조를 씁니다. **렌더된 mp4 하나가 여러 플랫폼으로 자동으로 팬아웃(fan-out, 부채처럼 퍼짐)** 됩니다.

`publish.mjs` 라는 발행 디스패처(교통정리 담당) 하나가 진입점입니다. 여기에 영상 하나를 물려주면, 유튜브·틱톡·인스타그램·페이스북 네 곳에 동시에 올라갑니다. 쓰레드는 조금 특별합니다. 쓰레드는 영상 없이 **순수 텍스트로 뿌리 글을 올리고 답글을 릴레이로 잇는** 방식이라, 별도 워커(일꾼 프로그램)가 담당합니다.

> ⚙️ **더 깊이** — `app/saju-shorts/scripts/publish.mjs` 는 "무엇을·어디에" 올릴지만 라우팅하고, 실제 업로드는 플랫폼별 워커에 위임합니다. 유튜브는 자체 워커(`scripts/youtube/upload.mjs`), 틱톡·인스타·페북은 **공통 통합 API 워커** 하나를 거칩니다. 쓰레드만 `noVideo: true` 플래그로 mp4 게이트를 우회하고 `scripts/threads/upload.mjs` 가 텍스트 뿌리글 + 대댓글 체인을 발행합니다. 명령 한 줄(`--platforms all`)이면 전 플랫폼에 한 번에 나갑니다.
>
> 지금 같은 구조를 새로 짠다면, 이 통합 워커를 **직접 만들지 않는 길**도 있습니다. 여러 채널을 한자리에 묶어 주는 발행 창구(예: uplika)를 에이전트에 붙이면, 워커 코드 대신 "이 영상 올려줘" 한 줄로 같은 자리를 채웁니다. 사주시바는 이 구조를 먼저 만든 쪽이라 워커를 직접 들고 있는 것뿐이에요.

여기서 중요한 단어가 **organic(오가닉)** 입니다. 광고비를 한 푼도 안 내고, 플랫폼의 자연 노출(추천·탐색 탭·팔로워 피드)로만 도달하는 방식을 오가닉이라고 합니다. 돈 주고 밀어 넣는 광고(paid)의 반대말이죠. 사주시바는 이 오가닉에 승부를 걸었습니다. 이유는 곧 나옵니다.

[C] 맨 아래 — 웹 첫 결제

영상을 본 사람이 프로필의 **링크인바이오**(프로필에 걸어둔 대표 링크)를 누르면 사주시바 웹으로 들어옵니다. 랜딩 페이지를 거쳐, 무료로 자기 운세를 확인하고, 결국 **첫 결제**에 다다릅니다. 이게 퍼널의 종착점이자 이 프로젝트가 최종적으로 원하는 행동입니다.

그런데 바로 이 마지막 한 걸음, "무료로 구경한 사람이 처음으로 돈을 쓰는 순간"이 가장 어렵고 가장 잘 새는 지점입니다. 그래서 사주시바는 이 지점에만 별도의 설계 문서를 파고, 기능 리스트를 우선순위로 나눠 붙였습니다. 다음 섹션이 그 이야기입니다.

첫 결제 트리거 — 가장 잘 새는 구멍을 막는 설계

먼저 다시 한번 강조합니다. 지금부터 설명할 첫 결제 기능들은 전부 **아이디어 단계(status: idea)의 설계 의도** 입니다. "이렇게 하면 결제율이 오를 것이다"라는 가설이지, "이렇게 했더니 올랐다"는 실측 결과가 아닙니다.

⚙️ **더 깊이** — 근거 기획서 `wiki/features/첫결제-트리거-개선.md` 의 프론트매터에 `status: idea` , `priority: P0` , `goal: 획득+전환` 이라고 명시돼 있습니다. 문서 하단 "검증 필요 (가설)" 섹션에는 "하드→소프트 게이트 전환이 실제 첫결제율을 올리는지 A/B", "워터마크 공유 카드의 오가닉 재유입률" 등이 **아직 검증 안 된 질문** 으로 남아 있습니다.

기획서는 우선순위 라벨(P0 → P1 → P2)로 트리거를 정렬했습니다. P0가 가장 급한, 지금 당장 막아야 하는 구멍입니다.

P0 — 대표 상품의 "하드 게이트"를 허문다

레퍼런스 사이트의 구조를 뜯어보니 치명적인 누수 지점이 보였습니다. 릴스를 보고 처음 들어온 **콜드 트래픽**(브랜드를 전혀 모르는 낯선 방문자)에게, 아무 결과도 안 보여준 채 **결과가 0%인 상태에서 돈부터 요구** 하는 구조였던 겁니다. 이걸 "하드 게이트"라고 부릅니다. 문이 딱 잠겨 있고, 열려면 무조건 결제하라는 방식이죠.

낯선 사람 입장에서 생각해보세요. 방금 재밌는 영상 하나 보고 호기심에 들어왔는데, 아직 이 서비스가 뭘 해주는지 하나도 못 봤는데, 대뜸 "990원 결제하세요"가 뜹니다. 대부분 그냥 나갑니다. 이게 콜드 트래픽 최대 누수 지점입니다.

사주시바의 해법: 990원짜리 대표 상품(사주·궁합)을 **개인화 티저 + 블러(뿌옇게 가림)** 로 통일합니다. 즉 결과를 아예 안 보여주는 게 아니라, "당신의 애정운은... (뿌옇게 가려진 부분)"처럼 **일부를 개인화해서 보여주고, 핵심만 흐리게 가리는** 방식입니다. 사람은 자기 이야기가 조금이라도 보이면 나머지가 궁금해서 못 견딥니다. 문을 완전히 잠그는 대신, 문틈으로 안을 살짝 보여주고 "더 보려면 여기"로 유도하는 겁니다.

P1 — 첫 결제 0원, 그리고 공유를 결제 '앞'으로

P1에는 두 가지 장치가 있습니다.

첫째, 첫 결제 0원 / 무료 "개껌" 1개 지급. 신규 유입자에게 첫 한 건은 공짜로 줍니다. 사주시바에서 유료 콘텐츠를 여는 재화 단위가 **개껌(🦴)** 인데(도사 시바가 강아지니까요), 이 개껌을 처음 온 사람에게 하나 쥐여줍니다. "일단 한 번 공짜로 써보세요"의 심리입니다. 결제라는 행위 자체의 심리적 문턱을 0원으로 낮춰서, 일단 "결제 버튼을 눌러본 경험"을 만들어 주는 거죠.

⚙️ **더 깊이** — 이건 근거 없는 발상이 아니라 검증된 벤치마크에서 왔습니다. 딥리서치 문서(`wiki/sources/deep-research-릴스수익화.md`)의 검증 통과 항목: "Daum 운세 '첫결제 0원' 프로모 실제 운영(1ID 최초 1회, 궁합·타로 포함 121개)"이 있고, 재화 충전 모델은 "포스텔러 = 현금 코인 충전 → 콘텐츠별 소비"가 한국 시장의 검증된 벤치마크로 인용됐습니다. 다만 이 문서도 "단 2014년, 지표 없음"이라고 조건을 달아 놓았습니다.

둘째, 공유를 결제 '앞'으로 옮긴다. 이게 가장 영리한 한 수입니다. 경쟁사는 공유 기능을 **결제 뒤에** 잠가 놨습니다. 돈을 내야 결과를 공유할 수 있죠. 그러면 어떻게 될까요? 결제 안 한 사람(대다수)은 공유를 못 하니까, 바이럴(입소문) 루프가 아예 차단됩니다.

사주시바는 반대로 갑니다. **무료 궁합 점수 카드에 워터마크(브랜드 표식)를 붙여서, 결제 전에 즉시 공유할 수 있게** 합니다. 무료로 나온 내 궁합 점수를 친구에게 바로 보낼 수 있고, 그 카드에는 사주시바 로고가 박혀 있습니다. 친구가 그걸 보고 또

들어오고, 또 공유하고... **무료 바이럴 루프**가 돕니다. 광고비 0원으로 콘텐츠가 스스로 번식하는 구조죠.

P2 — 가격 앵커링과 간편결제

P2는 상대적으로 급하지 않은 최적화입니다. **가격 앵커링**(높은 가격 팩을 먼저 보여줘서 그다음 팩이 싸 보이게 만드는 심리 기법)과 **간편결제 최소 마찰**(결제 단계를 원탭으로 줄이기)입니다. 9,900원 팩을 "개껌당 최저가"로 프레이밍하는 식이죠.

> ⚙️ **더 깊이** — 가격 앵커링도 딥리서치 검증 항목에서 왔습니다: "가격앵커링(높은 티어 먼저 → 타깃이 싸 보임) + 월환산 프레이밍(Mojo +~~845%). — RevenueCat". 다만 이 +~~845% 수치는 사주시바가 아니라 인용 출처의 사례일 뿐, 사주시바의 실측이 아닙니다.

정리하면 첫 결제 트리거는 **하드 게이트를 소프트 게이트로(P0) → 결제 문턱을 0원으로(P1) → 공유를 앞으로 당겨 바이럴(P1) → 가격 심리 최적화(P2)** 의 순서로 설계됐습니다. 전부 "경쟁사가 이렇게 막혀 있으니, 우리는 이렇게 열자"는 반면교사에서 나온 가설입니다.

★ 원리 — 왜 이렇게 만들었나

여기가 이 챕터의 심장입니다. 개별 장치를 다 봤으니, 이제 그 밑에 깔린 **세 가지 원리** 를 봅니다. 사주시바가 왜 굳이 이렇게 만들었는지의 이유죠.

① 제작비 → 0	② 공식 API만	③ 첫결제 마찰 0
코드로 렌더하니 콘텐츠가 공짜에 수렴 → organic 도달을 **물량으로**	봇 인게이지먼트=섀도우밴. 주제별 세그먼트 계정 분산으로 안전한 발행량 유지	광고 CAC 대신 organic + 첫결제 문턱 제거에 베팅

노린 효과(가설): 저비용 유입 · 브랜드 반복 노출 · 바이럴 루프 — **아직 검증 중**

원리 ① — 콘텐츠 제작비를 0에 수렴시켜, 도달을 힘으로 밀어붙인다

오가닉 마케팅의 냉정한 진실은 이겁니다. **영상 한 편이 뜰지 안 뜰지는 아무도 모른다.** 100편을 올리면 98편은 조용히 묻히고 2편이 터집니다. 그런데 그 2편이 어느 것일지는 미리 알 수 없어요. 그럼 답은? **많이 올리는 것**입니다. 확률 게임이니까 시행 횟수를 늘리는 거죠.

문제는 사람이 영상을 만들면 제작비(시간·돈)가 편당 크게 들어서 "많이"가 불가능하다는 겁니다. 여기서 CH4의 영상 공장이 마케팅 무기로 돌변합니다. **코드로 렌더하면 편당 제작비가 0에 수렴합니다.** 그러면 무식하게 많이(brute-force, 힘으로 밀어붙이기) 찍어서 오가닉 도달의 확률 게임을 정면으로 돌파할 수 있습니다.

이게 첫 번째 원리입니다. **제작비를 0으로 만들어서, "많이 올려서 뭐가 터지는지 본다"는 전략을 실제로 실행 가능하게 만든 것.** 데이터 주도로 반복 생성되는 스코어카드 포맷이었기 때문에 자동화가 가능했다는 점을 기억해 두세요. (CH4에서 봤듯, 1회성·비정형 콘텐츠는 이렇게 못 찍어냅니다.)

원리 ② — 공식 API로만 발행한다 (봇 금지)

두 번째 원리는 "안 걸리는 법"입니다. 자동으로 대량 발행하다 보면 유혹이 생깁니다. "좋아요도 자동으로 눌러주고, 팔로우도 자동으로 하면 더 빨리 크지 않을까?" 이걸 **봇 인게이지먼트**라고 하는데, 사주시바는 이걸 **명확하게 금지** 했습니다.

이유는 **섀도우밴(shadow ban)** 입니다. 플랫폼이 "이 계정 봇 같은데?"라고 판단하면, 계정을 대놓고 정지시키는 대신 **조용히 노출을 죽여버립니다.** 본인은 정상적으로 올리는 것 같은데 아무한테도 안 보이는 상태가 되는 거죠. 오가닉 도달에 전부를 건 전략에서 섀도우밴은 사형선고입니다.

그래서 사주시바는 **플랫폼 공식 API를 통한 정식 발행만** 합니다. 자동화는 하되, 규칙을 지키는 자동화만 하는 거죠. 그런데 여기서 딜레마가 생깁니다. 한 계정으로 너무 많이 올리면 그것도 스팸으로 걸립니다(스로틀링). 이 딜레마의 해법이 아주 우아합니다. **한 계정에 몰아서 올리지 않고, 주제별(세그먼트) 계정 여러 개로 나눠 각 계정이 플랫폼이 허용하는 안전한 발행량 안에서만 움직이게 한다.**

> ⚙️ **더 깊이** — 딥리서치 검증 항목: "공식 API 발행만 안전. 봇 인게이지먼트=섀도우밴." 띠별·연애·취업·궁합처럼 주제를 쪼갠 세그먼트 계정을 여러 개 두면, 각 계정은 플랫폼 정책이 허용하는 안전한 발행량을 지키면서 전체 도달은 합쳐서 늘릴 수 있습니다. 이게 다섯 플랫폼(그리고 그 안의 여러 세그먼트 계정)으로 팬아웃하는 구조적 이유입니다.

즉, **"많이 올리되(원리①), 플랫폼 규칙 안에서 올린다(원리②)"** 가 짝을 이룹니다. 대량 발행과 계정 안전은 보통 상충하는데, 세그먼트 계정 분산이 둘을 화해시킵니다.

원리 ③ — 광고 CAC 최적화 대신, 오가닉 + 첫 결제 마찰 제거에 베팅

세 번째 원리는 "돈을 어디에 안 쓰기로 했나"입니다. 보통 스타트업은 **광고를 돌리고 CAC(고객 획득 비용)를 최적화** 합니다. "광고비 1만 원 써서 고객 한 명 데려오는데, 이걸 8천 원으로 줄이자" 같은 게임이죠. 이 게임은 돈이 계속 들어갑니다. 광고를 끄면 트래픽도 끊깁니다.

사주시바는 이 게임을 **안 하기로** 했습니다. 대신 두 가지에 베팅합니다.

1. **오가닉(광고비 0, 한계비용 거의 0)** — 원리①②로 만든 무료 도달. 영상 한 편 더 올리는 데 드는 추가 비용이 거의 0이니, 광고와 달리 "끄면 끊기는" 구조가 아닙니다.
2. **첫 결제 마찰 제거** — 광고비로 사람을 더 데려오는 대신, **이미 온 사람이 결제까지 가는 길을 매끄럽게** 만드는 데 공을 들입니다(앞의 P0~P2 트리거 전부).

한 문장으로: **"사람을 더 많이 사 오는(광고) 대신, 공짜로 오게 만들고(오가닉), 온 사람이 안 새게 막는다(첫 결제 마찰 제거)."** 벤치마크로 삼은 것도 이 방향의 검증 사례들입니다 — 다음(Daum) 운세의 '첫 결제 0원' 프로모, 포스텔러의 코인 충전 모델.

★ 노림수 — 무엇을 노렸나 (전부 가설)

원리를 실행하면 어떤 효과가 날 것으로 **노렸는지**, 다시 강조하지만 **전부 가설로** 정리합니다.

- **조직적인 오가닉 유입을 노렸다.** 팔로워 0에서 시작하니 화려함보다 **일관성**을 우선했습니다. 세 개의 스코어카드 쇼츠를 **주 3편, 월·수·금 아침 9시**에 자동 발행하도록 스케줄을 박았습니다. "뜨는 한 방"이 아니라 "꾸준히 쌓는 리듬"에 건 겁니다.

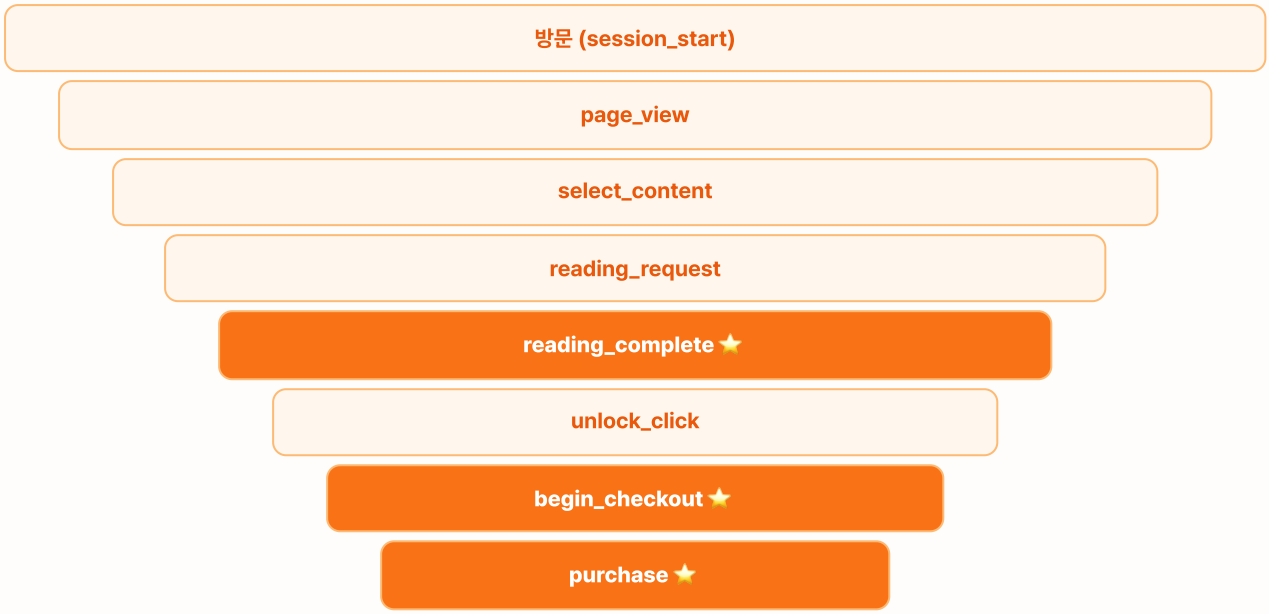
  > ⚙️ **더 깊이** — `shorts-scheduler`의 `cadence.json`에 세 컴포지션이 각각 `weekday: mon / wed / fri`, `time: 09:00`, `period: weekly`, `platforms: all`로 등록돼 있습니다. 스케줄러가 이 스펙대로 자면서 발행합니다(무인화 상세는 CH6). "월·수·금 아침"은 감이 아니라 설정 파일에 못박힌 사실입니다.

- **저비용 CAC를 노렸다.** 광고비가 0이니, 고객 획득 비용을 구조적으로 낮게 가져가려는 베팅입니다. 다만 실제 CAC가 얼마인지는 **아직 모릅니다**(뒤 정직성 섹션 참고).
- **브랜드의 반복 노출을 노렸다.** 도사 시바 마스코트가 **모든 프레임에 등장** 하고, 모든 캡션 끝에 성장 CTA(구독/팔로우 유도)가 붙습니다. 같은 캐릭터를 다섯 플랫폼에서 주 3편씩 계속 보게 만들어, "사주 = 도사 시바"라는 연상을 시간을 두고 각인시키려는 노림수입니다.
- **무료 바이럴 루프를 노렸다.** 앞의 P1(공유를 결제 앞으로)이 노리는 건 콘텐츠가 스스로 번지는 구조입니다. 광고 없이 사용자가 사용자를 데려오는 순환이죠.

이 노림수들이 **실제로 통했는지는 이 챕터가 대답할 수 없습니다.** 대답하려면 데이터가 필요하고, 데이터를 만들려면 측정 장치가 필요합니다. 그래서 사주시바는 측정 프레임을 먼저 깔았습니다.

측정 프레임 — 노림수를 데이터로 검증하려는 장치

가설을 세웠으면 검증해야 합니다. "노렸다"를 "됐다/안 됐다"로 바꾸려면 숫자가 필요하죠. 사주시바는 GA4(구글 애널리틱스4)를 웹에 심고, 퍼널을 8단계로 계측했습니다.



GA4 8단계 퍼널 · 가장 크게 빠지는 단계를 자동으로 짚어 개선 가설로 (⭐ = 핵심 이벤트)

8단계 퍼널은 이렇게 흘러갑니다. 낯선 사람이 들어와서(session_start) → 홈에 도착하고(page_view) → 메뉴를 고르고(select_content) → 해석을 요청하고(reading_request) → **무료 결과를 보고(reading_complete)** → 잠긴 부분을 누르고(unlock_click) → **결제를 시작하고(begin_checkout)** → **결제를 완료(purchase)** 하는 여정입니다. 각 화살표마다 사람이 얼마나 빠져나가는지(이탈률)를 재면, "어느 구간에서 가장 많이 새는가"를 콕 집을 수 있습니다.

그리고 **3개의 핵심 비율**로 건강 상태를 요약합니다.

- **활성화율** = 무료 결과 도달 ÷ 세션 (들어온 사람 중 몇 %가 실제로 결과를 봤나)
- **결제의도율** = 결제 시작 ÷ 무료 결과 도달 (결과 본 사람 중 몇 %가 결제하려 했나)
- **결제전환율** = 결제 완료 ÷ 결제 시작 (결제 시작한 사람 중 몇 %가 끝까지 갔나)

⚙️ **더 깊이** — 사주시바 전용 GA4 속성을 새로 파서 웹에 심었습니다. 유입은 **UTM 5채널**(youtube·tiktok·instagram·threads·facebook)로 태깅해 릴스 트래픽이 GA4에서 "direct"로 뭉치지 않게 채널별로 분리합니다. 결제는 브라우저 없는 서버 콜백에서 확정되기 때문에, 서버가 직접 **Measurement Protocol**로 `purchase` 를 GA4에 쏩니다. 이때 `client_id` 를 유지해서 **"유튜브에서 들어온 사람 → 첫 결제"까지 채널 어트리뷰션이 끊기지 않게** 하고, `transaction_id = billId` 로 클라이언트와 서버의 이중 집계를 자동 제거(dedup)합니다.

이 측정 프레임이 있으면 자동화가 한 바퀴 더 돕니다. `ga4-report` 라는 스킬(자동 업무 매뉴얼)이 매주 이 지표를 읽어서 **주간 리포트 + 개선 가설**을 생성합니다. "이번 주 최대 누수는 무료 결과 → 언락 구간. 원인 가설: 페이월 카피가 약함. 다음 실험: 카피 A/B"처럼요. 사람은 그 가설 중 하나를 골라 실험하고, 다음 주 리포트로 전후를 비교합니다. **측정 → 가설 → 실험 → 재측정**의 개선 루프죠.

⚙️ **더 깊이** — `ga4-report` 스킬은 GA4 MCP로 `run_funnel_report` · `run_report` · `run_conversions_report` 를 호출해 `outputs/reports/ga4-weekly-YYYY-Www.md` 를 생성하도록 설계돼 있습니다. 채널별 유입·8단계 이탈·3핵심비율·체류·매출을 표로 뽑고, §6에서 "최대 이탈 지점 + 원인 가설 2~3개 + 다음 실험 제안"을 합성합니다. 다만 스킬 문서 스스로 "첫 의미있는 리포트는 **프로덕션 배포 + 데이터 1주 누적 후**"라고 못 박아 뒀습니다.

아이디어는 어디서 오나 — reels-finder와 threads-forms

콘텐츠를 대량으로 찍어내는 것과, **무엇을** 찍을지 정하는 건 다른 문제입니다. 사주시바는 아이디어 소싱(어디서 훅을 가져올까)도 스킬로 만들어 뒀습니다.

reels-finder — 유튜브에서 사주/운세 영상을 훑되, 단순 조회수순이 아니라 채널 체급에 비해 유난히 터진 **"떡상" 신호를 자체 기준으로 점수화**해 줄을 세웁니다. 그렇게 찾은 영상에서 **포맷과 훅만 차용** 하고, 정작 데이터는 사주시바 자체 만세력으로 채웁니다. 남의 콘텐츠를 베끼는 게 아니라, "왜 이게 터졌나"의 구조만 학습하는 거죠.

threads-forms — 바이럴 쓰레드 글의 **구조 뱅크(라이브러리)** 입니다. 잘 터진 쓰레드의 뼈대(훅 유형·전개·마무리)를 모아 재사용하되, 마무리에 부르는 행동만큼은 원본 그대로 두지 않고 **사주시바의 목적(무료 확인·팔로우)에 맞게 바꿔 답니다.** 남의 글을 베끼는 게 아니라 구조만 빌리고, 독자에게 시키는 행동은 우리 것으로 채우는 거죠.

여기서도 [원리①]과 같은 철학이 보입니다. **껍데기(검증된 바이럴 구조)는 빌려오고, 알맹이(데이터·CTA)는 우리 것으로 채운다.** 경쟁사 역설계로 스펙을 발견하는 CH3의 태도가 마케팅에서도 그대로 반복됩니다.

⚠️ 정직한 고백 — 이건 "성과 낸 기계"가 아니라 "베팅"이다

이제 이 챕터에서 가장 중요한 이야기를 하겠습니다. 지금까지 원리도 정교하고, 노림수도 그럴듯하고, 측정 장치까지 갖췄습니다. 그런데 **아직 아무것도 증명되지 않았습니다.**

- **핵심 KPI가 확립되지 않았습니다.** CAC(고객 획득 비용)·전환율·리텐션(재방문율) — 이 셋 다 신뢰할 만한 출처로 확립된 수치가 없습니다. 딥리서치 문서조차 이걸 "오픈 퀘스천"으로 남겨 뒀습니다.

  ⚙️ **더 깊이** — 딥리서치 문서(`deep-research-릴스수익화.md`)는 25개 주장 중 14개만 검증 통과, 11개는 반증으로 인용 금지 처리했습니다. 반증 목록에는 "포스텔러 유료 전환율 8%", "AstroClub $90K MRR" 같은 솔깃한 수치가 전부 들어 있습니다. 이유: "전부 마케팅 에이전시 자기보고 단일출처. **핵심 KPI(CAC·리텐션·전환율)는 신뢰 가능한 출처로 확립되지 않음 = 오픈 퀘스천.**" 미해결 질문으로 "한국 사주앱 첫결제 전환·리텐션·ARPPU·CAC 실측치?", "릴스 조회 → 랜딩 → 첫결제 퍼널 전환율?" 등이 그대로 남아 있습니다.

- **GA4 실데이터도 아직 안 쌓였습니다.** 측정 프레임은 다 깔았는데, 정작 데이터가 흐르질 못하고 있습니다. 두 가지 블로커 때문입니다. **(1) Vercel 자동배포가 약 40시간 중단** 돼 프로덕션에 최신 코드(GA 계측 포함)가 안 올라갔고, **(2) GA4 MCP 인증(ADC)이 아직 안 됐습니다.** 이 둘이 풀려야 데이터가 흐르기 시작하고, 데이터가 1주 쌓여야 첫 의미 있는 리포트가 나옵니다.

> ⚙️ **더 깊이** — 설계 문서(`2026-07-05-analytics-funnel-improvement-design.md`) "오픈 이슈 / 선결 블로커": "Vercel 자동배포 ~40h 중단 — 이게 풀려야 0단계 이후 전부 시작 가능", "GA4 MCP ADC 미인증 — 자동리포트 엔진의 전제". 즉 개선 루프가 아직 첫 바퀴를 못 돌린 상태입니다.

그래서 정직하게 말하면, 이 마케팅 자동화는 **"성과를 낸 기계"가 아니라 "데이터를 만들어서 가설을 검증하려고 돌리기 시작한 베팅"** 입니다. 그리고 바로 이 점이 오히려 이 케이스스터디의 진짜 가치입니다. 대부분의 마케팅 사례는 성공한 뒤에 "이래서 됐다"를 그럴듯하게 되짚습니다(사후 서사). 사주시바는 **검증 전 상태를 그대로 보여줍니다** — 어떤 가설을, 왜 세웠고, 어떻게 측정해서 확인할지의 설계도 전체를요.

첫 결제 트리거가 `status: idea` 인 것도, GA4 리포트가 아직 안 나온 것도, KPI가 오픈 퀘스천인 것도 — 이건 프로젝트의 약점이 아니라 **"정직하게 검증 중"이라는 증거** 입니다. 클로드코드로 마케팅을 자동화한다는 게 "버튼 누르면 돈이 나온다"가 아니라, **"가설을 빠르고 싸게 세우고, 검증 장치까지 코드로 깔아서, 될지 안 될지를 데이터로 판정하는 시스템을 만드는 일"** 이라는 걸 이 챕터가 보여준 셈입니다.

그래서 뭐가 대단한가

한 발 물러서서 보면, 개인이나 작은 팀이 하기엔 말이 안 되는 일을 해냈습니다.

- **콘텐츠 제작비를 0에 수렴시켜** 오가닉 도달의 확률 게임을 힘으로 밀어붙일 수 있게 만들었습니다(원리①). 사람이 영상 편집을 했다면 애초에 불가능한 전략입니다.
- **대량 발행과 계정 안전이라는 상충을** 세그먼트 계정 분산 + 공식 API 발행으로 화해시켰습니다(원리②).
- **광고비 0원**의 오가닉과 첫 결제 마찰 제거에 베팅해, "끄면 끊기는" 광고 의존에서 벗어난 구조를 노렸습니다(원리③).
- 그리고 무엇보다, **가설을 세운 즉시 그것을 검증할 측정 프레임(GA4 8단계·3비율·주간 자동 리포트)까지 같이 깔았습니다.** 마케팅을 "감"이 아니라 "실험 시스템"으로 만든 것이죠.

이 모든 걸 별도의 마케팅팀·데이터팀 없이, 스킬과 스케줄러와 측정 계측으로 코드화했다는 점 — 그게 대단한 지점입니다. 완벽해서가 아니라, **한 사람이 마케팅 공장 전체를 설계도부터 검증 장치까지 세울 수 있게 됐다** 는 사실이 대단한 겁니다.

지금까지 CH4에서 "콘텐츠를 어떻게 자동으로 만드는가"를, CH5에서 "그걸로 무슨 마케팅을 노리는가"를 봤습니다. 그런데 이 모든 게 **사람이 자는 동안에도 굴러가려면** 누군가는 정해진 시간에 스위치를 눌러줘야 합니다. 사람 없이 스위치를 누르는 그 장치, 다음 챕터에서 볼 **3중 하네스(무인화의 뼈대)** 이야기입니다.

CH6 — 무인화 = 3중 하네스: 자는 동안 돌아가는 공장

월요일 아침 9시, 아무도 없는데 영상이 올라간다

상상해 보세요. 월요일 아침 9시. 아직 잠에서 덜 깬 시간입니다. 그런데 그 순간, 맥북 어딘가에서 조용히 프로그램 하나가 깨어납니다. "오늘이 무슨 요일이지? 아, 월요일이네. 월요일엔 띠별 운세를 올리기로 했지." 프로그램은 이번 주 만세력(사주 계산의 근거가 되는 달력) 데이터를 새로 뽑고, 그 숫자를 영상 템플릿에 끼워 넣고, 영상을 렌더링해서, 유튜브·틱톡·인스타그램·페이스북·쓰레드 5개 채널에 한꺼번에 올립니다. 그리고 "발행 완료"라고 장부에 한 줄 적고 다시 잠듭니다.

그 사이 사람은 아무것도 하지 않았습니다. 클릭 한 번, 프롬프트 한 줄 없었습니다.

이게 바로 이 챕터의 주제입니다. **자는 동안에도 돌아가는 무인 시스템은 실제로 어떻게 생겼는가.** 그리고 여기서 놀라운 반전이 하나 있습니다. 이 매일 도는 루프 안에는 **AI(LLM)가 단 한 줄도 없습니다.** AI가 매일 아침 일하는 게 아니라, AI는 이미 오래전에 이 공장을 다 지어 놓고 빠져나갔거든요. 그 이야기를 지금부터 풀어 보겠습니다.



자는 동안에도 스스로 도는 무인 콘텐츠 공장 — 도사 시바는 자고 있고, 컨베이어는 계속 영상을 찍어낸다.

⚙️ **더 깊이** — 이 챕터가 해부하는 실제 파일들: 스케줄러 설계는 `.claude/skills/shorts-scheduler/SKILL.md`, 운행 스케줄은 `cadence.json`, 중복 방지 장부는 `ledger.json`, 실제로 매일 도는 오케스트레이터 코드는 `app/saju-shorts/scripts/scheduler/run.mjs`, macOS 예약 실행 정의는 `com.sajushiba.shorts-scheduler.plist` 입니다. 전부 사주시바 실제 운영 저장소(`saju_workspace`)에 있는 파일입니다.

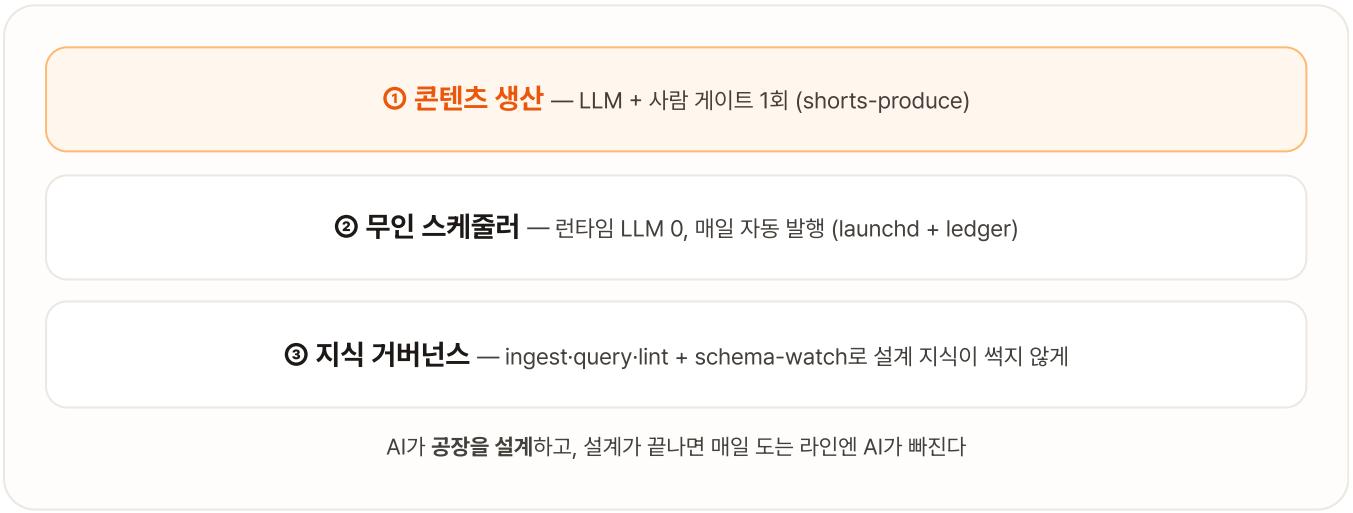
하나가 아니라 셋 — 겹쳐 있는 세 개의 자동화

먼저 큰 그림부터 잡고 갑시다. 흔히 "자동화"라고 하면 뭔가 하나의 거대한 버튼을 떠올리기 쉽지만, 사주시바에는 성격이 전혀 다른 **세 겹의 자동화가 포개져** 돌아갑니다. 세 개를 구분하지 못하면 시스템이 그냥 "마법"처럼 보이지만, 나눠 보면 각자 할 일이 명확한 세 명의 일꾼입니다.

① 콘텐츠 생산 하네스 (사람 게이트 있음). CH4에서 자세히 봤던 `shorts-produce` 오케스트레이터입니다. 새 영상의 "종류(템플릿)"를 처음 만들어 낼 때 도는 라인이죠. 기획 → 대본 → 데이터 채우기 → 목소리 → **브랜드 검수(사람이 딱 한 번 확인)** → 렌더 → 캡션 → 발행. 여기엔 사람의 눈이 반드시 한 번 들어갑니다. 새로운 걸 만들 때는 판단이 필요하니까요.

② 무인 스케줄러 (런타임에 LLM 전혀 없음). 이미 만들어진 템플릿을, 매주 정해진 요일에, 사람 없이 반복 발행하는 라인입니다. 이 챕터의 주인공이죠. 여기엔 AI도, 사람도 없습니다. 오직 결정론적인 계산과 정해진 규칙만 있습니다.

③ 지식 거버넌스 (설계 지식이 썩지 않게 지킴). 제품과 콘텐츠를 만든 "설계 지식" 그 자체를 관리하는 라인입니다. `/ingest` · `/query` · `/lint` 세 커맨드와 `schema-watch`가 위키를 살아 있게 유지합니다. 눈에 잘 안 보이지만, 이게 없으면 위 두 하네스의 근거가 시간이 지나며 낡고 어긋나기 시작합니다.



세 개를 굳이 나눈 이유가 있습니다. CH2에서 우리는 **"워크플로우 vs 하네스"**라는 구분을 배웠죠. 워크플로우는 요리 레시피(사람 말로 된 흐름과 재료 릴레이)이고, 하네스는 그 요리를 매일 실제로 해내고, 태운 접시를 잡아내고, 같은 접시를 두 번 내지 않는 "주방"이라고요. 이 챕터는 바로 그 주방이 완성되는 지점입니다. 지금부터 두 번째 하네스, 무인 스케줄러의 내부를 열어 보겠습니다.

두 번째 하네스 딥다이브 — LLM이 없는 스케줄러

심장: 매일 세 번 깨어나는 알람

무인화의 심장은 macOS에 내장된 `launchd`라는 예약 실행 장치입니다. 윈도우의 "작업 스케줄러", 혹은 그냥 아주 정확한 알람시계라고 생각하시면 됩니다. 사주시바는 이 알람을 **매일 오전 9시, 11시, 오후 2시(한국시간)** 세 번 울리도록 맞춰 놨습니다.

왜 하루에 세 번일까요? **9시가 주력**입니다. 운세 콘텐츠는 하루를 시작하는 아침에 소비되니까요. 그럼 11시와 2시는 뭘까요? **캐치업(catch-up), 즉 보험**입니다. 만약 9시에 어떤 이유로 발행이 실패했거나 맥북이 잠들어 있었다면, 11시가 다시 시도하고, 그래도 안 됐으면 2시가 또 시도합니다. 반대로 9시에 이미 잘 올라갔다면, 11시와 2시는 "어? 이미 올라갔네" 하고 아무것도 하지 않고 지나갑니다. 이 "이미 올라갔는지 확인하는" 장치가 뒤에 나올 장부(ledger)입니다.

> ⚙️ **더 깊이** — `com.sajushiba.shorts-scheduler.plist`의 `StartCalendarInterval`에 `Hour 9`, `Hour 11`, `Hour 14`가 배열로 박혀 있고, `RunAtLoad`는 `false`입니다(등록하는 순간 곧바로 쏘지 않고, 다음 정각을 기다린다는 뜻). 실행 시 `WorkingDirectory`는 `app/saju-shorts`, 실제로 실행되는 명령은 `node scripts/scheduler/run.mjs` 한 줄뿐입니다. 표준출력·에러는 스킬 폴더의 `logs/launchd.{out,err}.log`로 남습니다. 켜고 끄는 건 `scheduler.sh start` / `stop`이 `launchctl load` / `unload`로 처리합니다.

두 개의 문서가 두뇌 역할을 한다 — cadence.json과 ledger.json

알람이 울려서 프로그램이 깨어났다고 해봅시다. 이 프로그램은 "오늘 뭘 해야 하지?"를 어떻게 알까요? 사람이 매번 알려주지 않는데도요. 답은 **두 개의 JSON 문서**입니다. 이 둘이 스케줄러의 두뇌입니다.

첫 번째, `cadence.json` **— 운행 시간표(기계용 스케줄).** 사람이 읽는 `posting-cadence.md`(쇼츠별 성격·주기를 정리한 정본)를 기계가 읽을 수 있게 옮겨 적은 표입니다. 여기엔 딱 세 줄이 들어 있습니다.

- **월요일 09시** → 띠별 운세 (`ZodiacFortuneShorts`)
- **수요일 09시** → 운 강한 사주 TOP5 (`WeeklyStrongSajuShorts`)
- **금요일 09시** → 재물운 사주 (`SajuPaljaShorts`)

주 3편, 월·수·금으로 요일을 분산해 놓은 겁니다. 프로그램은 깨어나면 오늘 요일을 보고 "월요일이니 cadence 표에서 월요일 줄만 골라내자"라고 판단합니다. 요일이 안 맞으면 그냥 조용히 종료하고요.

두 번째, `ledger.json` **— 발행 원장(중복 방지의 단일 진실).** 이건 회계 장부와 똑같습니다. 뭔가를 발행할 때마다 "언제, 무엇을, 어디에 올렸는지"를 한 줄씩 적습니다. 여기서 핵심은 **키(열쇠)**입니다. 각 기록의 열쇠는 `기간 + 콘텐츠 종류`로 만들어집니다. 예를 들어 "2026년 27주차 + 띠별 운세"가 하나의 열쇠죠. 이 열쇠에 "완료(done)" 도장이 찍혀 있으면, 스케줄러는 그 콘텐츠를 **건너뜁니다.**

바로 이 장부 덕분에, 하루에 알람이 세 번 울려도 **중복 발행이 정확히 0건**입니다. 9시에 발행하면서 장부에 "27주차 띠별 운세 = 완료"라고 적어 두면, 11시와 2시에 깨어난 프로그램은 장부를 보고 "이미 완료네, 스킵" 하고 지나갑니다. 반대로 9시가 실패해서 장부에 완료 도장이 없으면, 11시가 다시 시도해 캐치업합니다. 아주 단순하지만 강력한 안전장치예요.

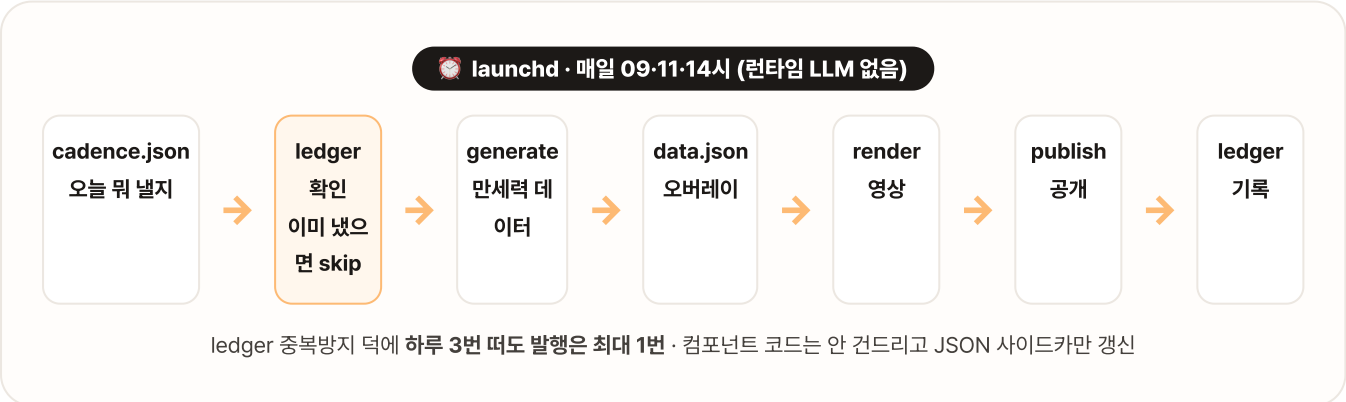
⚙️ **더 깊이** — 기간 키는 주간 콘텐츠면 ISO 주차(예: `2026-W27`), 만약 매일 콘텐츠라면 날짜 그 자체로 계산됩니다(`run.mjs`의 `periodKey`·`isoWeek` 함수). 중복 판정은 `ledgerHasDone(ledger, period, composition)`이 `status === "done"`인 레코드를 찾아 true면 skip. 발행에 성공하면 `recordLedger`가 `{period, composition, publishedAt, status:"done", note:"scheduler auto"}`를 append 하고, 실패하면 같은 자리에 `status:"fail"`을 적은 뒤 macOS 알림(`osascript`)을 띄웁니다. 캐치업이 성공하면 기존 fail 레코드는 지우고 done으로 덮어씁니다.

실제로 매일 도는 코드 — run.mjs의 6단계

이제 진짜 몸통입니다. `run.mjs`는 알람이 울릴 때마다 실행되는 **순수 오케스트레이터**입니다. 여기서 "순수"라는 말이 중요합니다. 이 파일의 첫 줄 주석에 대놓고 이렇게 적혀 있어요: `LLM 無`(LLM 없음). AI에게 뭘 물어보지도, 생성하지도 않습니다. 그냥 정해진 순서대로 도구들을 호출하는 지휘자일 뿐입니다.

오늘 발행할 콘텐츠가 정해지면(요일 확인 + 장부 확인 통과), 각 편마다 이 6단계가 순서대로 돕니다.

1. **① 데이터 생성** — 만세력 계산기(`shorts-gen.ts`)를 불러 이번 주 실제 사주 데이터를 뽑습니다. "이번 주 재물운 강한 일주 TOP5는 누구인가" 같은 결과가 순수 계산으로 나옵니다.
2. **② data.json 오버레이 기록** — 방금 뽑은 데이터를 영상 템플릿 옆에 있는 `data.json`이라는 작은 파일에 덮어씁니다. (이 파일이 이 시스템의 숨은 핵심인데, 바로 아래에서 따로 설명합니다.)
3. **③ 쓰레드 대댓글 체인 갱신** — 쓰레드(Threads)용 원문 + 대댓글 글타래를 새 데이터로 갈아 끼웁니다.
4. **④ 목소리(TTS) — 스킵** — 여기가 재밌습니다. 나레이션 대사는 매주 안 바뀌거든요("이번 주 재물운 강한 사주는..."). 대사가 그대로면 목소리도 그대로니까, **기존에 만들어 둔 오디오 파일을 재사용**하고 새로 만들지 않습니다. 비용도 시간도 아끼는 거죠.
5. **⑤ 렌더** — Remotion(코드로 영상을 그리는 도구)이 새 데이터가 반영된 세로 영상(mp4)을 만들어 냅니다.
6. **⑥ 발행 → 장부 기록** — `publish.mjs`가 5개 플랫폼에 **공개(public)**로 한꺼번에 올리고, 성공하면 장부에 "완료" 도장을 찍습니다. 실패하면 "실패" 기록 + 알림을 남기고, 다음 캐치업 슬롯이 처리하도록 넘깁니다.



ledger 중복방지 덕에 **하루 3번 떠도 발행은 최대 1번** · 컴포넌트 코드는 안 건드리고 JSON 사이드카만 갱신

⚙️ **더 깊이** — `generate` 는 `tsx` 로 `shorts-gen.ts` 를 실행해 `{ data, thread[] }` 를 반환합니다(`data` =만세력 결과, `thread[]` =쓰레드 원문+대댓글 배열). `writeDataOverlay` 는 `src/remotion/<compositionFolder>/data.json` 에 기록하고, 렌더는 `./node_modules/.bin/remotion render <CompId> out/<CompId>.mp4` , 발행은 `node scripts/publish.mjs --composition <id> --platforms all --privacy public` 입니다. 나레이션을 정말 바꿔야 할 때만 `--tts` 플래그로 `gen-voice.mjs` 를 돌려 목소리를 재생성합니다. `--dry-run` 은 발행·장부 기록 없이 리허설만, `--force` 는 요일·중복을 무시하고 특정 편을 강제 발행합니다(테스트용).

숨은 핵심 — 사이드카 파일 data.json / voice.json

여기서 이 시스템이 왜 그렇게 안정적인지를 결정하는 아주 영리한 설계 하나를 짚고 넘어가야 합니다. 바로 **`data.json` 과 `voice.json` 이라는 "사이드카" 파일**입니다.

영상 템플릿의 **본체 코드**(레이아웃, 색깔, 마스코트 위치, 애니메이션 등 React/Remotion 컴포넌트)는 콘텐츠 생산 하네스에서 사람과 AI가 함께 공들여 만든 결과물입니다. 이 코드는 한번 완성되면 **매주 절대 건드리지 않습니다.** 그럼 매주 바뀌는 숫자(이번 주 운세 점수, 날짜, 랭킹)는 어디로 들어갈까요? 바로 옆에 붙은 작은 `data.json` 파일로만 들어갑니다.

즉, 무인 스케줄러가 매주 하는 일은 "본체는 그대로 두고, 옆에 붙은 작은 데이터 봉투만 새 내용으로 갈아 끼우는 것"입니다. 이것이 바로 **"LLM이 만든 정교한 템플릿"과 "사람 없이 도는 데이터 갱신"의 이음새**입니다. 오토바이 본체(사이드카가 붙은)는 그대로 두고, 옆 좌석(사이드카)에 탄 승객만 매주 바꿔 태우는 셈이죠. 본체는 검증이 끝난 안정된 물건이라 건드릴 이유가 없고, 매주 바뀌는 건 데이터 봉투 하나뿐이라 실수할 여지가 극히 적습니다.

⚙️ **더 깊이** — `data.json` 은 컴포넌트의 `captions[0].data` 를 오버라이드하는 오버레이입니다. 스케줄러는 이 파일만 새로 쓰고 `src/remotion/**` 컴포넌트 코드는 손대지 않습니다. `voice.json` 도 같은 사이드카 패턴으로, 나레이션이 불변인 한 재생성되지 않습니다. 폴더 이름(`HeavenChosenLotto` , `ZodiacLuckyNumbers` 등)이 과거 흔적처럼 보여도, 실제 콘텐츠는 사이드카 데이터로 완전히 분리돼 있어 "로또" 같은 단어는 데이터·대본 어디에도 나오지 않습니다.

핵심 통찰 — AI는 공장을 짓고, 다 지으면 빠진다

이제 이 챕터에서 가장 중요한 문장을 말할 차례입니다. 매일 도는 스케줄러 루프에 **왜 LLM이 없는가.** 이걸 이해하면 무인화의 진짜 원리를 손에 쥐게 됩니다.

답은 이렇습니다. **LLM의 역할이 전부 "셋업(초기 설정)" 단계로 밀려났기 때문입니다.**

- **무엇을 만들지** (어떤 운세 콘텐츠가 대중에게 먹힐지)
- **어떤 레이아웃으로** (마스코트 위치, 자막이 콘텐츠를 가리지 않게)
- **브랜드에 맞는지** (팔레트, 톤, 로또 단어 배제)

이런 "판단"이 필요한 일은 전부 사람과 AI가 **셋업 때 한 번** 처리합니다. CH4의 콘텐츠 생산 하네스(사람 게이트 포함)가 바로 이 셋업 라인이죠. 그리고 셋업이 끝나면, 매주 반복되는 실무에는 판단이 필요 없습니다. 필요한 건 오직 **정확한 계산(만세력)**과 **정해진 규칙(요일·중복 방지)**뿐이니까요. 그래서 매일 도는 런타임 루프에서 AI를 완전히 빼버릴 수 있는 겁니다.

이걸 한 문장으로 요약하면 이렇습니다:

> **"AI가 공장을 설계하고, 설계가 끝나면 빠졌다."**

이 설계가 왜 그렇게 좋은지 세 가지로 정리해 볼게요.

- **안정적이다.** 매번 AI에게 물으면 매번 답이 조금씩 다릅니다(같은 질문에도 표현이 달라지죠). 하지만 결정론적 계산은 언제 돌려도 똑같은 답을 냅니다. 만세력 계산은 100% 재현 가능하고, 요일·중복 규칙도 항상 동일하게 작동합니다. 무인으로 며칠·몇 주를 방치해도 삐끗할 이유가 없습니다.
- **저렴하다.** 매일 세 번, 여러 편을 발행하는데 AI 호출이 0건이면 그만큼 운영비가 안 듭니다. 목소리(TTS)도 대사가 그대로면 재사용하니 추가 비용이 없고요.
- **무인이다.** 판단할 게 없으니 사람이 지킬 필요가 없습니다. 사람은 "공장을 지을 때"와 "공장을 껐다 켤 때"만 개입하면 됩니다.

이것이 클로드코드로 만든 자동화의 성숙한 형태입니다. AI를 매일 굴리는 게 아니라, **AI로 규칙과 도구를 만들어 두고 AI를 빼는 것.** 초심자는 "매일 AI가 알아서 해주는 시스템"을 상상하지만, 진짜 잘 만든 무인 시스템은 오히려 "AI가 필요 없게 만든 시스템"입니다.

세 번째 하네스 — 지식이 썩지 않게 지키는 거버넌스

이제 세 번째 자동화입니다. 이건 영상도, 발행도 아닌, **"이 모든 걸 만든 설계 지식 자체"를 관리하는 하네스**입니다. 눈에 잘 안 띄지만, 없으면 시간이 지나며 조용히 무너지는 종류의 것이죠.

사주시바 저장소는 지식을 세 층으로 엄격히 나눕니다(Karpathy의 LLM 위키 패턴).

- `raw/` **= 불변 원본.** 사람이 자료를 떨어뜨려 놓기만 하는 곳. **수정·삭제·이동 금지.** AI는 읽기만 합니다. 경쟁사 캡처, 리서치, 릴스 레퍼런스 같은 날것의 재료 창고죠.
- `wiki/` **= LLM 전담.** AI가 원본을 소화해서 페이지로 정리하고, 링크로 엮고, 갱신하는 정돈된 지식 공간. 경쟁사 분석, 기능 기획, 콘텐츠 패턴, 원본 요약이 여기 삽니다.
- `outputs/` **= 파생물.** 리포트, 쿼리 답변, 리프레시 대기열 같은 AI 산출물 전용. (코드 제품은 여기 두지 않습니다.)

이 세 층을 살아 있게 유지하는 게 세 개의 커맨드입니다.

- `/ingest` — `raw/`의 새 원본을 읽어 `wiki/`의 알맞은 페이지로 통합합니다. 모순이 발견되면 기존 내용을 지우지 않고 **나란히 병기**합니다. 양방향 링크(A가 B를 참조하면 B도 A를 참조)도 챙기고요.
- `/query` — 위키에 자연어로 질문하면 **출처(citation)를 달아** 답합니다. 그리고 좋은 답변은 그냥 흘려보내지 않고 다시 위키에 페이지로 **되심습니다("good answers filed back into the wiki").** 답할 때마다 지식이 한 겹씩 두꺼워지는 거죠.

- `/lint` — 위키의 건강검진입니다. 서로 모순되는 서술, 아무 데서도 링크되지 않은 **고아 페이지**, 클릭하면 깨지는 **죽은 링크**, 필수 정보(frontmatter) 누락을 찾아냅니다. 단, **직접 고치지 않고 제안만** 합니다. 수정은 사람 승인 후에요.

그리고 이 세 커맨드 각각의 끝자락에는 `schema-watch` **라는 감시자**가 자동으로 붙습니다. 이 감시자의 역할은 하나입니다. **"규칙(스키마)과 실제 현실이 어긋나기 시작했는가"**를 감지하는 것. 예를 들어 문서엔 없는 새 종류의 페이지가 자꾸 생기거나, 죽은 참조가 반복되거나, 같은 위반이 3번 넘게 나오면, `schema-watch` 는 그 신호를 `outputs/reports/refresh-queue.md` 라는 대기열에 차곡차곡 쌓습니다. 신호가 임계치를 넘으면 "리프레시 권장"이라고 한 줄 알립니다.

이게 바로 **설계 지식이 썩지 않게 지키는 하네스**입니다. 제품 코드는 만들면 끝이 아니라, 그 코드를 왜 그렇게 만들었는지에 대한 "지식"이 시간이 지나며 낡습니다. 가격이 바뀌고, 도구가 바뀌고, 경쟁사가 바뀌는데 문서가 그대로면 어느 순간 문서는 거짓말을 하기 시작하죠. `/ingest` · `/query` · `/lint` + `schema-watch` 는 그 부패를 막는 면역 체계입니다.

> ⚙️ **더 깊이** — `schema-watch` 는 실제 리프레시를 하지 않습니다. 감지만 하고 대기열에 쌓을 뿐이에요. 실제 정리는 별도 커맨드(`/wiki-refresh`)가 사람 승인 아래 처리합니다. 감지 → 대기열 → 승인 후 일괄 정리로 역할을 분리해, 자동화가 문서를 제멋대로 뜯어고치는 사고를 원천 차단합니다. `raw/` 불변 규칙과 "직접 수정 금지, 제안만"이라는 lint 규칙이 같은 철학이죠.

정직한 경고 — "start" 버튼은 진짜 발행이다

무인 시스템을 소개하면서 반드시 정직하게 짚어야 할 게 있습니다. 이 자동화는 **연습이 아니라 실전**이라는 점입니다.

스케줄러를 켜는 명령(`scheduler.sh start`)을 실행하는 순간부터, 시스템은 **매주 월·수·금에 실제로 5개 플랫폼에 공개(public) 발행을 시작합니다.** 리허설이 아니라 진짜 세상에 나가는 콘텐츠예요. 그래서 이 시스템의 스킬 문서에는 "**라이브 arming**(실탄 장전)"이라는 경고와 함께 **"사용자 확인 필수"**가 명시돼 있습니다. 켜기 전에 반드시 사람이 "정말 지금부터 진짜 발행합니다"를 확인해야 한다는 뜻이죠.

또 하나, 이 무인 시스템에는 물리적 전제가 있습니다.

- **맥북이 깨어 있어야 합니다.** 절전 모드로 잠들거나 로그아웃되면 그 시각의 발행은 건너뜁니다.
- **GUI 로그인 상태여야 합니다.** 영상을 그리는 Remotion이 화면이 있는 크롬(headful Chrome)을 필요로 하기 때문입니다.

빠지면 어떻게 될까요? 다행히 앞서 본 안전장치들이 받쳐 줍니다. 9시에 맥북이 잠들어 실패해도 11시·2시 캐치업이 다시 시도하고, 실패는 장부에 기록되며 알림이 뜹니다. 그래도 며칠씩 맥북이 꺼져 있으면 발행은 밀립니다. 그래서 "완전 무인"이라기보다 **"사람은 공장을 켜고 끄고, 가끔 잘 도는지만 확인하면 되는 시스템"**이라고 이해하는 게 정확합니다. 리스크를 숨기지 않고 이렇게 정직하게 아는 것이, 무인 시스템을 실제로 운영하는 사람의 자세입니다.

⚙️ **더 깊이** — `scheduler.sh start`는 plist를 `launchctl load`로 등록하고, `stop`은 `unload` 후 파일을 제거합니다. `RunAtLoad`가 `false`라 등록 즉시 쏘지 않고 다음 정각을 기다립니다. `status`로 현재 ON/OFF를 확인할 수 있고, `run --dry-run`으로 실제 발행 없이 리허설만 돌려볼 수 있습니다. 실패 알림은 `osascript` macOS 알림으로, 실행 로그는 `logs/`에 날짜별로 남습니다.

그래서 뭐가 대단한가

정리해 봅시다. 사주시바의 무인화는 하나의 마법 버튼이 아니라, 성격이 다른 **세 겹의 하네스**가 맞물린 시스템입니다.

- **콘텐츠 생산 하네스**가 판단이 필요한 일(새 템플릿 만들기)을 사람 게이트와 함께 처리하고,
- **무인 스케줄러**가 판단이 필요 없는 반복(매주 발행)을 LLM 없이 결정론으로 처리하고,
- **지식 거버넌스**가 이 모든 설계의 근거가 낡지 않도록 지켜냅니다.

그리고 이 셋을 관통하는 하나의 원리가 있죠. **판단은 셋업으로 몰고, 반복은 무인으로 뺀다.** AI를 매일 굴리는 대신 AI로 규칙과 도구를 만들어 두고 빠지는 것. 그래서 이 시스템은 안정적이고, 저렴하고, 자는 동안에도 돌아갑니다.

이게 CH2에서 던졌던 질문의 답입니다. 워크플로우(레시피)만으로는 매일 아침 요리가 저절로 나오지 않죠. 그 레시피를 **매일 실제로 요리하고(오케스트레이터), 태운 접시를 잡아내고(장부·캐치업), 같은 접시를 두 번 안 내는(중복 방지) 주방**이 있어야 비로소 무인 공장이 됩니다. 이 챕터에서 우리는 그 주방—오케스트레이터 + 계약(사이드카 데이터) + 스케줄러(launchd·cadence·ledger) + 거버넌스(ingest·query·lint)—이 실제로 어떻게 조립돼 돌아가는지 눈으로 확인했습니다.

가장 인상적인 건, 이 모든 게 **거대한 기술 조직이 아니라 한 사람의 맥북 한 대에서** 돌아간다는 사실입니다. 비전공자도 클로드코드로 "판단은 셋업에, 반복은 무인에"라는 원리만 손에 쥐면, 자는 동안 콘텐츠를 찍어내는 공장을 실제로 세울 수 있다는 얘기죠.

다음 챕터로

세 겹의 하네스까지 봤으니, 이제 남은 건 이 모든 걸 **당신의 프로젝트로 옮겨 심는 법**입니다. 다음 챕터에서는 사주시바에서 뽑아낸, 어디에나 이식할 수 있는 6단계 플레이북과 "어디서부터 시작할지"를 정리합니다.

CH7 — 비법 정리 + 적용 로드맵

여기까지 오시느라 고생 많으셨습니다. 지금까지 우리는 사주시바라는 실제 서비스가 어떻게 **하루 만에** 만들어졌는지, 그 서비스에 물을 주는 **영상 공장**과 **마케팅 자동화**가 어떻게 돌아가는지, 그리고 사람이 자는 동안에도 콘텐츠가 발행되는 **3중 하네스**가 어떻게 조립됐는지를 하나씩 뜯어봤습니다.

그런데 이 책의 진짜 목적은 "사주 서비스를 따라 만드세요"가 아닙니다. 사주는 그냥 **재료**였습니다. 이 마지막 챕터의 목표는, 앞의 여섯 챕터에서 흩어져 있던 방법들을 **어떤 프로젝트에도 그대로 얹을 수 있는 하나의 플레이북**으로 압축하는 것입니다. 사주가 아니라 여러분의 카페 예약 서비스든, 독서모임 관리 도구든, 부모님 가게 홍보 자동화든 — 똑같은 순서로 굴러갑니다.

그리고 이 챕터 끝에서, 저는 여러분에게 **오늘 당장 시작할 첫걸음** 하나를 손에 쥐어드리려 합니다. 거창한 서비스가 아니라, 아주 작은 것 하나요.

이 책 전체를 한 문장으로

사주시바를 만든 방식을 딱 한 줄로 줄이면 이렇습니다.

> **"머릿속 상상으로 코드를 짜지 않는다. 경쟁사에서 스펙을 발견하고, 코드보다 설계문서를 먼저 쓰고, AI에게 '무엇을·어떤 순서로·어디까지'를 문서 한 장으로 넘긴 뒤, 원자 단위 체크리스트를 100%까지 채우고, 반복되는 일은 매뉴얼(스킬)로 박아 결국 사람 없이 돌아가게 만든다."**

말이 길죠? 그래서 이걸 여섯 걸음으로 나눴습니다. 이게 오늘 여러분이 가져가실 **플레이북**입니다.

이식 가능한 6단계 플레이북

1. **경쟁사를 해부 → 스펙 발견**
   사주: 레퍼런스 사이트 딥크롤
2. **plan-first: 코드 전에 설계문서**
   사주: 설계문서 5종
3. **goal = 킥오프 프롬프트 1장**
   사주: 킥오프 한 장
4. **loop = 체크리스트 100%**
   사주: 54/54
5. **결정론 계산 ↔ AI 서술 분리**
   사주: 계산 폴더 ↔ 해석 폴더
6. **스킬 → 커맨드 → 하네스로 무인화**
   사주: 영상 공장 + 스케줄러

앞의 여섯 챕터에서 이미 하나씩 본 것들이라, 여기서는 이름표만 다시 붙입니다. 각 단계를 여러분 프로젝트의 재료로 바꿔 읽으시면 됩니다.

1단계 — 경쟁사를 해부해, 스펙을 '발견'한다

"이런 서비스 있으면 좋겠다"는 빈 상상에서 곧장 코드로 뛰어들지 않습니다. 이미 그 시장에서 돈을 벌고 있는 경쟁사를 끝까지 뜯어보고 기록하면, 만들 것의 목록이 **상상이 아니라 관찰에서** 나옵니다. 사주시바도 이 관찰 하나로 "복제할 것 3개 / 피할 실패 3개"를 손에 쥐고 출발했습니다(CH3).

2단계 — plan-first: 코드보다 설계문서를 먼저 쓴다

AI는 방향이 틀린 코드도 놀랍도록 빠르게 짜냅니다. 그래서 코드 전에 "무엇을·왜·어떤 순서로"를 사람 말로 먼저 적습니다. 잘못된 문단 한 줄 고치는 건 5초지만, 잘못 짜인 코드 수백 줄을 되돌리는 건 반나절이니까요. 사주시바는 코드 0줄 상태에서 설계문서 다섯 장을 먼저 썼습니다(CH3).

3단계 — goal: 킥오프 프롬프트 한 장으로 넘긴다

설계문서가 여러 장 있어도, AI에게는 **"자, 이제 시작하자"는 한 장**이 따로 필요합니다. 무엇을 만들지, 어떤 순서로, 어떤 규칙 안에서, 그리고 아직 안 정해진 질문은 무엇인지를 한 장으로 압축해 넘기면, 대화가 아무리 길어져도 AI가 이 북극성으로 돌아옵니다(CH3).

4단계 — loop: 원자 단위 체크리스트를 100%까지 채운다

큰 덩어리를 "됐다/안 됐다"를 한눈에 판정할 수 있을 만큼 잘게 쪼개고, 90%가 아니라 **100%**를 목표로 끝까지 돕니다. 남은 10%가 보통 진짜 문제들이거든요. 사주시바는 원자 항목 54개를 전부 채운 **54/54**라는 숫자로 완성을 정의했습니다(CH3).

5단계 — 결정론 계산 ↔ AI 서술을 물리적으로 분리한다

이 책에서 가장 값진 한 수입니다. 틀리면 안 되는 **사실 계산은 규칙 기반 코드**가 처리하고(같은 입력 = 항상 같은 출력), **말맛이 필요한 서술만 AI**에게 맡기도록 폴더 단위로 갈라놓습니다. AI가 사실을 지어낼 틈이 사라지고, 캐시로 비용이 줄고, AI가 죽어도 서비스는 삽니다. 자세한 구조는 CH3 §7에서 이미 봤습니다.

6단계 — 반복 업무를 스킬로 → 커맨드로 → 여러 개 엮어 하네스로

한 번 하고 마는 일과 **매번 반복하는 일**을 구분하고, 반복하는 일은 **스킬**(업무 매뉴얼)로 적어두고, 자주 부르면 **커맨드**(단축 지시어)로, 조각이 여러 개 엮이면 사람 없이 스스로 돌고·실패를 잡아내고·같은 일을 두 번 안 하는 **하네스**로 승격시킵니다. 사주시바의 영상 공장(CH4)과 무인 스케줄러(CH6)가 이 사다리의 끝 그림입니다.

여섯 단계를 가로지르는 습관 — 게이트와 질문

여기에 눈에 잘 안 띄지만 결정적인 습관 두 가지가 관통합니다. **매 단계 끝에 '통과 기준(게이트)'을 걸어 틀린 채로 다음 단계로 넘어가지 못하게 하는 것**(사주시바의 골든테스트가 그 예입니다), 그리고 **미결 사항은 AI가 임의로 정하지 않고 사람에게 되묻게 하는 것.** 문서를 어떻게 나누고 킥오프에 무엇을 담을지는 프로젝트마다 조금씩 다르겠지만, 방향은 위 여섯 줄이 전부입니다. 이 여섯 걸음과 두 가지 습관만 손에 쥐면, 여러분의 프로젝트도 충분히 끝까지 만들 수 있습니다.

그래서 이 여섯 단계가 대단한 이유는요 — 어느 하나 특별한 천재성이 필요 없다는 점입니다. 경쟁사 관찰, 문서 먼저 쓰기, 지시서 한 장, 체크리스트 채우기, 계산과 말 나누기, 반복 자동화 — 전부 **평범한 규율**입니다. 비전공자도 그대로 따라 할 수 있습니다. AI 시대의 실력은 "코드를 얼마나 잘 짜느냐"가 아니라 "이 규율을 얼마나 성실히 지키느냐"로 옮겨갔습니다.

어디서 시작할지 — 오늘의 첫걸음

여기까지 읽고 "좋은데, 이걸 다 하려니 막막하다" 싶으실 겁니다. 당연합니다. 그래서 딱 하나만 말씀드릴게요.

서비스부터 만들려고 하지 마세요. '작은 반복 업무 1개'부터 자동화해 보세요.



여러분이 매주 손으로 반복하는 일을 하나 떠올려 보세요. 이런 것들이요.

- 매주 월요일마다 여러 곳에 붙여넣는 정기 안내문 만들기
- 가게 신메뉴가 나올 때마다 SNS용 소개글 3종 쓰기
- 손님 문의를 정해진 톤으로 답장하기
- 엑셀에 쌓인 이번 주 매출을 요약 리포트로 정리하기

이 중 하나를 골라, AI에게 **"매뉴얼(스킬)"** 하나로 적어 두세요. "이런 걸 부탁하면, 이런 순서로, 이런 톤과 규칙으로 만들어라." 그게 여러분의 첫 스킬입니다. 완벽할 필요 없습니다. 한 번 써보고 어색하면 매뉴얼을 고치면 됩니다.

그걸 **자주 부르게 되면**, 짧은 한 마디로 부를 수 있는 **커맨드**로 승격시키세요. "그 주간 안내문 만들어줘"를 한 단어로요.

그리고 그런 조각이 **두세 개 엮이기 시작하면** — 예를 들어 "요약 리포트를 만들고 → 그걸 안내문으로 바꾸고 → 정해진 곳에 발행"까지 순서대로 이어지면 — 축하합니다, 여러분은 방금 **하네스**를 만든 겁니다. 사주시바의 영상 공장도 정확히 이렇게, 작은 조각들이 하나씩 붙어 큰 자동화가 됐습니다.

완벽함보다 '일단 배포'가 먼저입니다. 세상에서 제일 좋은 자동화는 "완벽하게 설계했지만 아직 안 켠 것"이 아니라 "허술하지만 오늘부터 돌아가는 것"입니다. 작게 시작해서 매주 조금씩 고치세요. 사주시바도 처음부터 3중 하네스가 있었던 게 아니라, 영상 한 편을 손으로 만들어 본 다음 그 과정을 매뉴얼로 굳히고, 그걸 스케줄러에 얹은 것뿐입니다.

> ⚙️ **더 깊이** — 이 사다리를 사주시바 파일에 대입하면 이렇습니다. **스킬 한 개** = `script-gen` (대본 쓰기 매뉴얼) 같은 조각 스킬 → **여러 조각을 순서로 묶은 오케스트레이터** = `shorts-produce` (8단계 릴레이) → **그 위에 얹은 무인 스케줄러** = `shorts-scheduler` (launchd 09·11·14시 + 원장 dedup). 여러분의 첫 사다리는 이보다 훨씬 작아도 됩니다. 조각 하나면 충분한 시작입니다.

정직한 마무리

마지막으로, 이 책에서 제가 계속 조심스럽게 말씀드린 부분을 다시 정직하게 짚고 끝내겠습니다.

사주시바의 마케팅 성과는 아직 검증 중입니다. 저비용으로 트래픽을 만들고, 5개 플랫폼에 반복적으로 브랜드를 노출하고, 첫 결제의 마찰을 없앤다는 **노림수(가설)**는 세워 뒀습니다. 측정 도구(GA4)도 붙여서 이제 막 데이터를 쌓기 시작했습니다. 하지만 "이 방식이 실제로 저비용에 고객을 데려온다"거나 "전환율이 몇 %다" 같은 **결론은 아직 없습니다.** 특히 이 책에서 언급한 첫 결제를 유도하는 장치는, 성과가 확인된 기능이 아니라 아직 **설계 의도(status: idea)** 단계의 아이디어입니다. 그건 앞으로 데이터로 검증하거나, 틀렸으면 뒤집어야 할 **가설**입니다. 저는 그걸 성과인 척 포장하지 않겠습니다.

그런데 바로 그 점이 이 책의 진짜 메시지입니다. **여러분이 가져가실 값진 것은 "사주로 돈 벌었다"는 결과가 아니라, '만드는 방식' 그 자체입니다.** 경쟁사를 관찰해 스펙을 발견하고, 설계를 먼저 쓰고, 지시서 한 장으로 넘기고, 체크리스트를 끝까지 채우고, 계산과 말을 나누고, 반복을 자동화하는 이 여섯 걸음은 — 사주가 잘되든 안 되든 **여러분의 다음 프로젝트에 그대로 남습니다.** 성과는 프로젝트마다 다르지만, 방식은 이식됩니다. 그게 이 방식의 힘이고, 여러분이 오늘 배워 가실 진짜 자산입니다.

이제 여러분 차례입니다. 매주 반복하는 그 귀찮은 일 하나를 떠올리셨나요? 오늘, 그걸 매뉴얼 한 장으로 적어 보세요. 완벽하지 않아도 괜찮습니다. 일단 켜세요. 그리고 매주 조금씩 고쳐 나가세요. 여러분이 상상만 하던 것이, 손으로 만질 수 있는 무언가가 되는 순간을 저는 진심으로 응원합니다.

바이브 메이커 · @vibemak2r