

지브의 작업실

나만의 AI 비서실 · 미리보기

나만의 AI 비서실

완성본에서 표지·목차와 대표 설명 페이지를 골라 묶은 미리
보기입니다.

미리보기 · 총 14쪽

전체 108쪽 중 선별한 원본 13쪽과 이 안내 1쪽입니다. 보조자료 파일은 포함하지 않습니다. 문제의 출발점부터 실제 구조·데이터 모델·검증 방법까지 확인하는 용도입니다.



나만의 AI 비서실

코덱스로 내 사업을 아는 비서와
프로젝트 팀을 만드는 법
지브의 작업실



목차

프롤로그 AI에게 일을 맡기고도 내가 쫓아다닌 이유

이 구조에는 각자의 일하는 방식이 들어가야 한다
그냥 하나씩 시키면 되지 않을까

1부 AI를 더 썼는데 왜 내가 더 바빠졌을까

1장 일 시킨 사람이 프로젝트 사이를 뛰어다녔다

2장 전달만 하는 비서는 필요 없었다

3장 역할과 규칙을 늘리기 전에

2부 내가 원한 것은 내 사업을 아는 비서였다

4장 지브라는 이름을 붙인 이유

5장 비서가 맡을 일과 프로젝트가 맡을 일

6장 사람이 마지막 결정을 남겨 둔 이유

3부 왜 코덱스였을까

프롤로그 AI에게 일을 맡기고도 내가 쫓아다니는 이유

처음에는 AI 프로젝트가 많아지면 내가 할 일이 줄어들 줄 알았다.

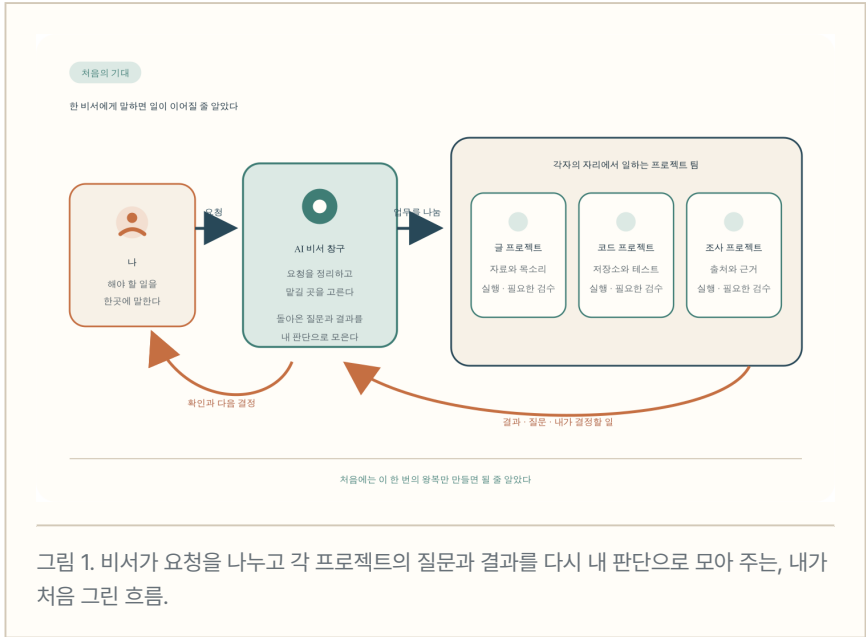
한 프로젝트에서는 글을 만들고, 다른 프로젝트에서는 코드를 고치고, 또 다른 곳에서는 자료를 조사했다. 각각의 AI는 자기 자리에서 꽤 많은 일을 해냈다. 문제는 그 사이였다. 어떤 결과가 나왔는지 확인하고, 그 결과를 다음 프로젝트에 설명하고, 빠진 부분을 찾아 다시 지시하는 사람은 나였다. 일을 시킨 사람이 프로젝트 사이를 뛰어다니는 배달원이 된 셈이다.

나는 혼자 일한다. 혼자 일하는 사람이 쓸 수 있는 시간과 집중력에는 분명한 한계가 있다. 그때 이런 생각을 했다.

개인 AI 팀을 만들면 내가 쓸 수 있는 능력의 범위를 넓힐 수 있을까.

내가 처음 머릿속에 그린 모습은 단순했다. 나는 비서 한 명에게만 일을 말한다. 비서는 요청을 정리해 글, 코드, 조사 프로젝트 가운데 알맞은 곳에 맡긴다. 각 프로젝트는 자기 자료와 파일 안에서 실행하고, 필요하면 검수까지 마친다.

일이 끝나면 결과와 질문이 다시 비서에게 모인다. 비서는 내가 확인하고 결정할 것만 추려 보여 준다. 이 구조라면 나는 프로젝트 사이를 돌며 내용을 옮기는 대신 목표와 마지막 판단에 집중할 수 있을 것 같았다.



그때는 이 한 번의 왕복만 만들면 될 줄 알았다.

막상 만들어 보니 실행하는 AI를 늘리는 것만으로는 이 흐름이 생기지 않았다. 누가 요청을 정리하고, 누가 각 프로젝트의 실제 상황을 확인하며, 누가 결과를 검수하고, 누가 마지막 결정을 하는지 정해 두지 않으면 결국 내가 다시 모든 연결을 맡게 됐다.

머릿속의 그림을 실제로 옮겨 처음 만든 비서는 단순했다. 내가 한 말을 받아 적당한 프로젝트에 보내고, 결과가 오면 다시 나에게 전달했다. 얼핏 비서처럼 보였다. 그러나 중요한 결과가 빠져도 통과시켰고, 검토가 필요한데도 완료라고 전했으며, 문제가 발견돼도 원래 작업자에게 재작업을 보내지 못했다. 나는 여전히 결과를 처음부터 읽고, 부족한 부분을 설명하고, 어느 프로젝트로 돌아가야 하는지 결정했다.

전달만 하는 비서는 내 일을 덜지 못했다. 전달은 자동화됐지만, 결과를 판단하고 다음 일을 잇는 몫은 여전히 내게 남았다.

시행착오의 규모

거의 4주치

제대로 된 구조를 찾는 시행착오에 쓴 Pro 사용량

Pro 요금제에서 사용량을 소진하고 초기화되는 과정을 네 번 겪었다. 네 번은 구조를 고친 횟수가 아니다.

내게 생긴 변화 업무판을 만든 뒤 맡긴 일·진행 중인 일·내 답이 필요한 질문·완료 결과가 한눈에 보이기 시작했다. 여러 대화창을 돌아다니는 일도 줄었고, 비서와 프로젝트 팀이 각자 몫을 맡는 작은 회사에 가까워졌다고 느꼈다. 연결을 프롬프트에 기대지 않고 프로그램이 말도록 바꾸자, 새 업무는 프로젝트 실행부터 결과 회수와 사용자 완료까지 한 흐름으로 이어지기 시작했다.

독자가 해볼 일 자기 업무 한 건에 시작·실행·질문·결과·다음 판단이 이어지는 순서를 그려 본다.

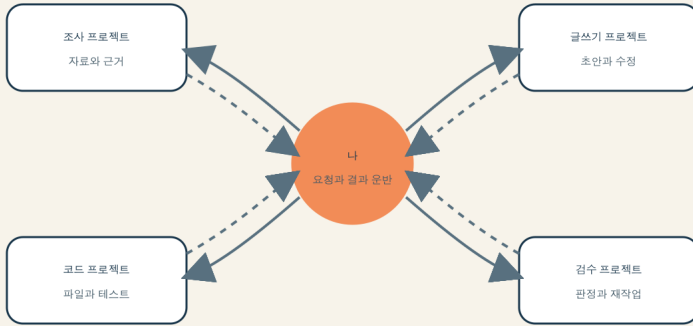
1장 일 시킨 사람이 프로젝트 사이를 뛰어다녔다

AI를 한 곳에서만 쓸 때는 운영 문제가 잘 보이지 않는다. 요청한 자리에서 답을 받고, 마음에 들지 않으면 바로 이어서 수정하면 된다. 하지만 사업에는 서로 다른 종류의 일이 있다. 글은 글의 자료와 목소리를 알아야 하고, 코드는 저장소와 테스트를 알아야 하며, 운영은 실제 계정 상태와 어떤 행동 전에 내 확인이 필요한지 알아야 한다. 일을 제대로 맡기려고 프로젝트를 나누는 순간부터 연결 문제가 생긴다.

내 하루에는 여러 개의 탭이 열려 있었다. 어느 대화에서는 시장 자료를 기다렸고, 다른 대화에서는 전자책 원고를 검토했으며, 또 다른 작업에서는 사이트 수정 결과를 확인했다. AI가 질문을 남기면 내가 찾아가 답했고, 결과가 나오면 다음 프로젝트가 이해할 수 있게 다시 요약했다. 검수에서 문제가 생기면 처음 요청과 결정 내용을 다시 찾아 정리했다.

실행 속도는 빨라졌는데 업무의 전체 속도는 내가 얼마나 잘 기억하느냐에 묶였다. 어떤 일을 먼저 시켰는지, 최신 결과가 어느 것인지, 어떤 수정은 반영됐고 어떤 것은 계획만 세웠는지, 누가 내 답을 기다리는지 모두 내 머릿속에 있었다.

AI가 늘었지만 연결은 사람에게 남았다



각 AI는 일했지만 끝난 결과와 다음 할 일은 내가 기억했다

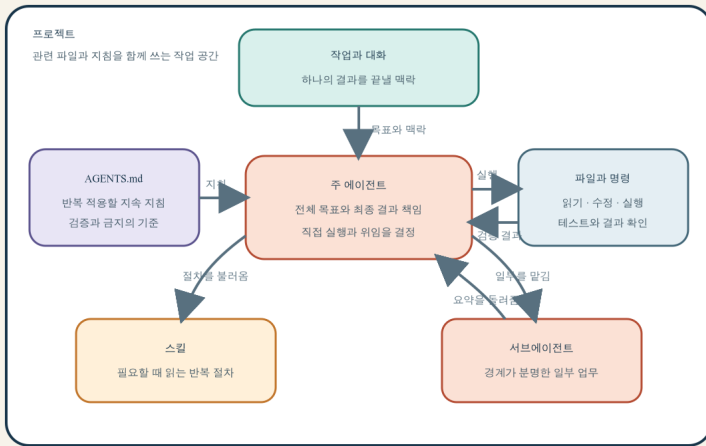
그림 2. 각 AI는 일했지만 요청·질문·결과의 연결은 사람에게 남아 있었다.

이전에는 이 문제를 AI의 이해력 부족이라고 생각하기 쉬웠다. 더 자세히 지시하거나 더 좋은 모델을 고르면 해결될 것 같았다. 물론 모델의 능력은 중요하다. 하지만 내가 겪은 병목은 다른 곳에도 있었다.

요청이 한 문장의 바람으로 남으면 실행자는 완료를 정의할 수 없다. 필요한 사실이 다른 프로젝트에 있으면 다시 묻게 된다. 결과가 원래 목표와 연결되지 않으면 사람이 어느 결과가 최신인지 찾아야 한다. “완료했습니다”라는 문장이 검증보다 먼저 오면 사용자는 다시 전부 읽어야 한다. 다음 행동을 AI가 마음대로 시작하면 일은 많아지지만 우선순위는 흐려진다.

내게 부족했던 것은 요청을 받는 곳, 각 프로젝트가 책임질 일, 결과를 확인하는 사람, 질문과 결과가 돌아오는 곳을 잇는 구조였다.

한 업무의 중심에는 주 에이전트가 있다



ZHIV와 업무판은 이 기능들을 사업 흐름에 맞게 연결한 자의 운영 설계다

그림 4. 한 업무의 중심에는 전체 목표와 결과를 책임지는 주 에이전트가 있다.

한 업무는 주 에이전트가 책임진다

먼저 에이전트부터 이해해야 한다. 에이전트는 목표를 받고 파일과 도구를 사용해 결과를 만드는 실행 담당이다. Codex에서 사용자의 요청을 직접 받아 처음 목표부터 최종 결과까지 책임지는 에이전트를 이 책에서는 주 에이전트라고 부른다.

주 에이전트는 목표와 완료 기준을 읽고, 프로젝트의 파일과 지침을 확인하며, 직접 처리할 일과 나눠 맡길 일을 결정한다. 서브에이전트가 돌려준 내용을 원래 목표에 맞게 합치고, 최종 결과와 검증 결과와 남은 일을 사용자에게 보고하는 책임도 주 에이전트에게 남는다.

전체 아키텍처

한 번 말하고, 책임은 다섯 계층으로 나눈다

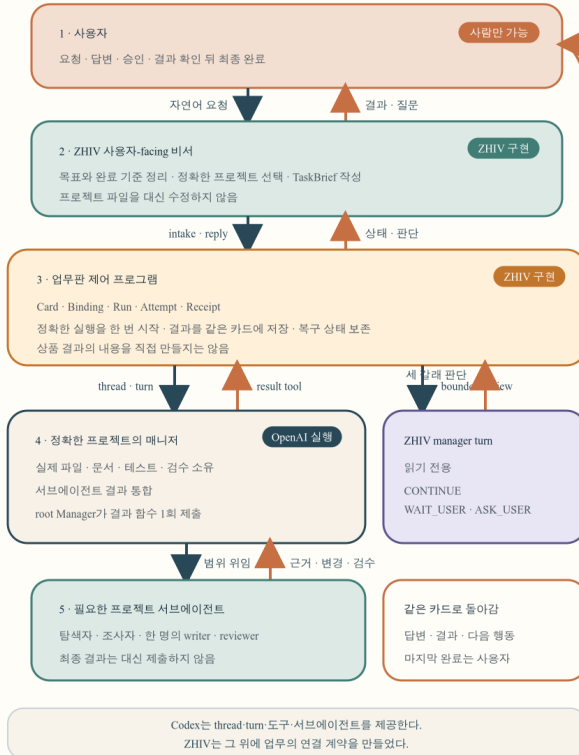


그림 6. 말은 지브에게 한 번 하지만, 내부에서는 비서·제어 프로그램·프로젝트 매니저·서브에이전트의 책임이 분리된다.

첫 번째는 사람이다. 하고 싶은 일과 사업의 우선순위를 말하고, 결과를 쓸지와 외부 행동을 실행할지를 결정한다.

업무란 내부 모델

카드 하나 뒤에는 다섯 개의 연결 기록이 있다

무엇을 연결하는가

Card · 업무의 기준점
목표 · 상태 · revision · 현재 요약 · 다음 행동
사람이 보고 판단할 한 업무

어디서 할까

Binding · 정확한 실행 자리
canonical root · project thread · result transport
카드와 정확한 프로젝트를 고정

지금 무엇을 할까

Run · 처리할 한 행동
PROJECT_START · PROJECT_CONTINUE
MANAGER_REVIEW · generation · claim token
카드당 active run 하나

실제 turn 시작

Attempt · 이번 실행 주인
thread · turn · generation · lifecycle
현재 파일을 쓸 한 실행

한 번 처리

Receipt · 처리 영수증
envelope · content hash · result · decision
같은 결과는 다시 처리하지 않음

사용자 화면

inbox ———— running ———— waiting_user ———— done

revision은 카드 변화 · generation은 실제 실행 세대 · stale 판단은 SUPERSEDED
카드당 active run 1개 · exact root당 writer turn 1개 · turn 확인 전 running 금지

실행이 어디까지 왔는가

QUEUED

claim + lease

CLAIMED

thread ID 저장

THREAD_READY

turn ID 저장

TURN_RUNNING

이때만 카드 running

결과 transaction

result + receipt + 사용자 결과
+ MANAGER_REVIEW를 함께 저장

RECONCILE_PENDING

known thread · turn만 제한 확인

RECOVERY_REQUIRED

불명확한 시작 · tool 등록 실패 · process 손실
자통 재실행하지 않음

그림 9. 카드·프로젝트 연결·실행·시도·결과 영수증을 나누면 화면 상태와 실제 실행 상태를 혼동하지 않게 된다.

카드(Card)는 사람이 기억하고 판단할 업무다. 제목, 목표, 완료 기준, 현재 요약, 다음 행동, 상태를 가진다. 같은 목표의 수정과 후속 요청은 가능하면 이 카드에 계속 붙는다.

정상 실행 순서

결과를 먼저 저장하고, 그다음 다음 행동을 판단한다



그림 10. 새 목표는 정확한 프로젝트의 새 실행으로, 같은 카드의 후속 요청은 같은 스레드의 다음 턴으로 이어진다.

이때 App Server 프로세스를 하루 종일 켜 두고 모든 프로젝트를 감시할 필요는 없다. 업무판에 처리할 실행이 생겼을 때 짧게 시작하고, 그 실행의 thread와 turn을 관찰해 결과를 저장한 뒤 종료할 수 있다. 비어 있는 업무판을 읽으려고 LLM을 깨우지 않아도 되고, 프로젝트마다 별도 폴링 AI를 둘 필요도 없다.

중첩 구조를 많이 만들면 모델이 필드를 빠뜨리거나 다른 모양으로 제출할 가능성이 커진다. 내가 쓰는 실제 계약에서는 위 일곱 필드를 모두 필수로 받고, 허용하지 않은 추가 속성은 거부한다. `documents` 는 최대 다섯 개다. 사용자 입력이 필요 없으면 `inputRequestKind` 는 `none` , `inputRequestPrompt` 는 빈 문자열이어야 한다. 자세한 로그 전체가 아니라 사용자가 열어 볼 결과만 돌려주기 위해서다.

영수증을 믿기 전에는 호출의 주인과 순서를 확인한다.

1. 결과가 카드와 연결된 정확한 `project thread`와 `turn`에서 왔는가.
2. 서브에이전트가 아니라 전체 업무를 책임진 `root 프로젝트 매니저`가 호출했는가.
3. 한 `turn`에서 결과 도구를 한 번만 호출했는가.
4. 결과 도구를 부른 뒤 파일 수정이나 다른 도구 실행이 없었는가.
5. `turn`이 `completed` 로 끝났는가.
6. 사람에게 보여 줄 마지막 답이 프로그램용 결과 데이터 덩어리가 아닌가.

호출이 생기면 App Server는 클라이언트에 `item/tool/call` 요청을 보내며 `threadId` , `turnId` , 도구 이름과 인자를 함께 준다. 클라이언트는 연결된 `turn`과 일곱 필드 `schema`를 확인하고 성공 여부를 응답한다. 이 요청 한 번만 보지 않고 도구 `item`의 시작·완료와 `turn`의 최종 `completed` 까지 맞아야 유효한 결과로 저장한다.

서브에이전트가 자기 조사 결과를 곧바로 업무판에 제출하지 못하게 한 이유도 여기에 있다. 조사자에게는 “완료”여도 프로젝트 전체 완료 기준은 아직 남아 있을 수 있다. 프로젝트 매니저가 결과를 합치고 검증한 뒤 한 번 제출해야 카드의 결과가 된다.

프로그램은 결과 데이터와 중복 방지 영수증, 사용자에게 보여 줄 결과, 지브의 다음 판단 예약을 한 번의 저장 작업으로 묶는다. 전부 저장되거나 전부 취소되게 해, 결과만 남고 다음 판단이 사라지는 반쪽 상태를 막는다.

단위 테스트는 상태 규칙 하나를 빠르게 확인한다. 잘못된 프로젝트 경로를 거부하는가, 카드에 활성 run을 두 개 만들지 않는가, 같은 root의 writer fence가 작동하는가, 중복 영수증이 하나로 처리되는가, 다른 해시는 충돌이 되는가, 오래된 revision 판단이 적용되지 않는가, 사용자 완료 의도가 한 번만 사용되는가를 본다.

runner 통합 테스트는 가짜 또는 통제된 App Server 흐름으로 thread와 turn의 수명주기를 확인한다. 시작부터 자연어 final과 result tool, turn 완료까지 이어지는 정상 흐름뿐 아니라 결과 도구가 없거나 두 번 불린 경우, 잘못된 turn과 서브에이전트 결과를 제출한 경우, 결과 뒤 도구가 더 실행된 경우, turn이 interrupted나 failed로 끝난 경우, 결과 도구 등록에 실패한 경우, 연결이 끊긴 뒤 자동 재실행하지 않는 경우를 넣는다.

실제 E2E는 피해가 적은 정확한 프로젝트에서 한 카드가 끝까지 이어지는지 본다. 자연스러운 프로젝트 대화가 보이고, 필요한 서브에이전트가 한 번 일하며, 프로그램용 결과 데이터가 사용자 답변에 노출되지 않고, 구조화 결과와 manager 판단이 한 번씩 생기며, 결과나 질문이 **내가 확인할 일**에 도착하고, 사용자가 완료할 수 있어야 한다.

장면	기대 결과	막아야 할 회귀
새 목표 접수	카드·binding·첫 run이 함께 생성	카드는 있지만 worker 없음
실제 turn 시작	thread·turn 저장 뒤 running	실행 없는 진행 표시
정상 결과 제출	자연어 설명과 receipt가 각각 한 번	프로그램용 결과 데이터 노출, 결과 누락
결과 중복 수신	기존 receipt 재사용	결과·manager 판단 중복
같은 식별자, 다른 내용	conflict로 정지	새 내용을 조용히 덮어씀