VIBE CODING PLAYBOOK

비개발자에서 개발자로

바이브코딩 실전 가이드 & 체크리스트

코드를 한 줄도 못 써도 괜찮습니다. AI와 함께 실제 서비스를 기획하고, 만들고, 세상에 공개하고, 운영하는 데 필요한 **순서 · 원칙 · 체크리스트**를 한 권에 담았습니다. 실전에서 검증된 방법론과 실제로 일어나는 함정들을 비개발자의 언어로 정리했습니다.



작성일 **2026년 7월 19일** · 대상 **코딩 경험이 없는 예비 창업자 · 기획자 · 1인 메이커**

목차

1	바이브코딩이란 — 비개발자가 개발자가 되는 새로운 길
2	바이브코딩 5대 원칙 — 성공률을 결정하는 순서
3	성장 로드맵 — STEP 0 ~ 6 단계별 가이드
4	기술 스택, 딱 필요한 만큼만 이해하기
5	AI에게 말하는 법 — 프롬프트 실전 가이드
6	구조(아키텍처)를 알아야 하는 이유 — 4레이어와 스위치 설계
7	함정 사전 — 실전에서 만나는 에러와 대응법
8	실서비스 운영 — 법무·결제·알림·리드타임
9	AI와 오래 일하는 법 — 검증·보안·백업·비용·멘탈
10	위험 구간 지도 — 전문지식이 꼭 필요한 7가지 순간
11	종합 체크리스트 — 단계별로 체크하며 진행
부록	비개발자를 위한 용어 사전

1 바이브코딩이란 — 비개발자가 개발자가 되는 새로운 길

낯선 단어가 나오면 맨 뒤 부록(용어 사전)을 참고하세요. 이 장은 판을 이해하는 장입니다.

바이브코딩: 코드를 치는 대신, 말로 만든다

바이브코딩(Vibe Coding)은 **AI에게 대화로 요구사항을 전달해 소프트웨어를 만드는 방식**입니다. 당신이 "이런 서비스를 만들고 싶어"라고 말하면, AI가 코드를 작성하고 화면을 만들고 에러를 고칩니다. 당신은 채팅창과 미리보기 화면만 보면 됩니다 — 폴더, 파일, 코드는 AI의 영역입니다.

그렇다고 "아무나 버튼 하나로 서비스가 나온다"는 뜻은 아닙니다. AI는 강력하지만, **무엇을 · 어떤 순서로 · 어떤 기준으로 시킬지**는 여전히 사람의 몫입니다. 이 가이드는 바로 그 부분 — 시키는 기술 — 을 다룹니다.

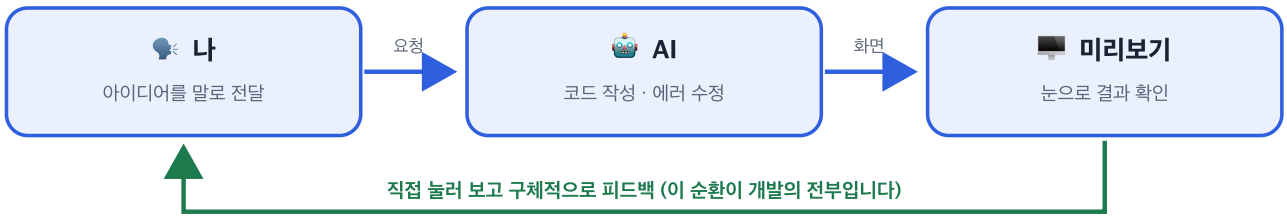


그림 1. 바이브코딩의 작동 구조 — 나는 말하고 확인하고, AI는 만들고 고친다

역할이 이렇게 바뀝니다

전통적 개발	바이브코딩	당신에게 남는 일
문법을 외우고 코드를 타이핑	AI가 코드 작성	요구사항을 명확한 말로 전달
에러 메시지를 해독하고 디버깅	AI가 원인 분석·수정	"고쳤는지" 직접 눌러서 확인
기술 선택·설계를 직접 공부	AI가 표준 구조 제안	선택지 중에서 의사결정
배포·서버 관리를 손으로	플랫폼 버튼으로 배포	공개 전 최종 점검과 운영

그런데 왜 "가이드"가 필요한가 — 바이브코딩의 함정

바이브코딩으로 만든 결과물의 흔한 평가는 "빠르지만 결함이 있다"입니다. 실패하는 프로젝트에는 공통 패턴이 있습니다:

- **순서 붕괴:** 기획 없이 시작 → 전체를 한 번에 요청 → 중간에 방향이 바뀌어 갈아엎음
- **검증 부재:** AI가 "완료했습니다"라고 하면 그대로 믿음 → 실제로는 안 돌아감
- **구조 부재:** 화면·데이터·서버 코드가 뒤엉켜, 어느 순간부터 "고칠수록 망가지는" 상태
- **운영 무지:** 만들기는 했는데 법무·결제·알림 같은 코드 밖의 일에서 좌초

✔ 이 가이드의 핵심 테제

비개발자의 개발 성공은 코딩 실력이 아니라 "순서와 규칙"에서 나온다. 올바른 순서(기획→화면→피드백→데이터→운영)를 지키고, 반복되는 실수를 규칙으로 만들어 기계에 맡기고, AI의 결과물을 직접 검증하는 습관을 갖추면 — 코드를 직접 쓰지 못해도 실서비스를 만들고 운영할 수 있습니다.

이 가이드를 읽는 법

- **지금 시작하는 사람:** 2장(원칙) → 3장(로드맵)을 읽고, 11장 체크리스트 A부터 체크하며 진행하세요.
- **이미 만들다가 막힌 사람:** 7장(함정 사전)에서 증상을 찾고, 6장(구조)으로 원인을 이해하세요.
- **출시를 앞둔 사람:** 8장(운영)·10장(위험 구간)과 11장 체크리스트 D~G가 당신 것입니다.
- 모든 장의 근거와 "왜"는 본문에 있고, 실행 항목은 11장에 모여 있습니다.

2 바이브코딩 5대 원칙

수많은 프로젝트의 성패를 가른 것은 재능이 아니라 이 다섯 가지 순서와 습관이었습니다. 먼저 두 경로를 비교해 보세요 — 출발점은 같습니다.

❌ 실패하는 경로 — "다 만들어줘"

아이디어 → 전체를 한 번에 요청 → 대량의 코드 → "생각과 다른데…" → 갈아엎기 반복

✔ 성공하는 경로 — "하나씩 확인하며"

아이디어 → 기획 대화 → 화면 1개 → 피드백 → 확장 → 데이터 → 출시

그림 2. 같은 아이디어, 다른 결말 — 차이는 재능이 아니라 순서

원칙 1 코딩보다 대화가 먼저다

좋은 바이브코딩 환경은 첫 요청에서 **절대 바로 코딩하지 않습니다.** 최소 1턴 이상 기획 대화로 아이디어를 구체화하고, "이대로 진행할까요?"라는 확인을 거친 뒤에야 작업을 시작합니다.

사용자가 할 일: AI가 확인 질문을 하면 귀찮아하지 말고 성실히 답하세요. 여기서 아낀 5분이 뒤에서 5시간의 재작업을 막습니다. AI가 질문 없이 바로 만들기 시작하면 오히려 멈추고 "먼저 방향부터 확인하자"고 요구하세요.

원칙 2 화면 먼저, 데이터베이스는 나중에 (Screen First, DB Later)

처음부터 회원가입·DB·결제를 다 만들면 피드백할 대상이 없는 채로 복잡도만 쌓입니다. 검증된 순서는 **임시(목) 데이터로 화면부터** → 눈으로 보고 피드백 → 방향이 확정되면 그때 실제 데이터 연동입니다. 이유는 단순합니다: **사람은 눈에 보이는 것에만 제대로 피드백할 수 있습니다.**

사용자가 할 일: "일단 임시 데이터로 화면부터 보여줘"가 기본 자세입니다. 실데이터 전환은 뒤의 원칙 4에서 한 번에 처리합니다.

원칙 3 한 번에 한 화면씩

"쇼핑몰 전체 만들어줘"는 실패 확률이 가장 높은 요청입니다. 첫 단계에서는 **랜딩/홈 1개 화면만** 만들고, 피드백을 받은 뒤 다음 화면으로 넘어가세요. 화면 하나 = 피드백 한 사이클입니다.

사용자가 할 일: 요청을 화면 단위로 쪼개세요. 그리고 매 화면마다 브라우저에서 직접 눌러 보고 구체적으로 피드백하세요. "예쁘게 해줘"보다 "버튼이 너무 작고, 제목이 두 줄로 깨져요"가 백 배 낫습니다.

원칙 4 실데이터 전환은 별도의 공식 관문으로

화면이 어느 정도 갖춰지면 **임시 데이터를 실제 데이터베이스로 전환하는 단계**를 하나의 공식 관문으로 진행하세요. 잘 갖춰진 환경에서는 "리팩토링 해줘" 같은 한마디가 이 관문을 엽니다: 구조 정리 → 임시 데이터 정규화 → **"회원/상품/주문 중 어떤 걸 실제로 연동할까요?"** 확인 → 선택한 것만 실제 DB로 전환. 이때 관리자 패널도 실제 데이터와 연결됩니다.

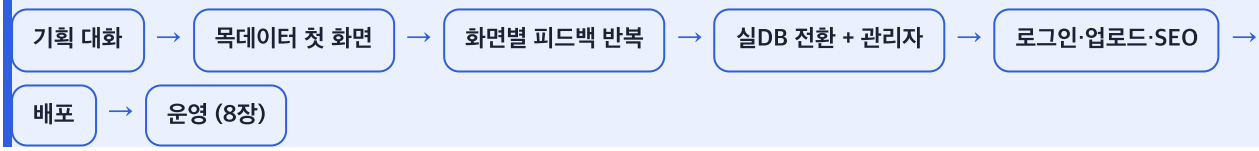
사용자가 할 일: 연동 여부를 물으면 도메인별로 선택할 수 있습니다("회원만 먼저"). 전부 연동할 필요 없습니다 — 운영에 필요한 것부터. 그리고 이 관문을 건너뛰고 "그때그때 조금씩 DB를 붙이는" 방식은 구조를 뒤엉키게 하는 지름길이니 피하세요.

원칙 5 반복되는 실수는 말이 아니라 규칙으로 막는다

AI에게 "그거 하지 말라고 했잖아"를 반복하는 것은 소모전입니다. AI는 대화가 길어지면 앞의 지시를 잊습니다. 해법은 두 가지입니다: ① **프로젝트 지침 문서에 규칙을 적어 두기**(AI가 매번 읽는 문서), ② **검사 규칙(ESLint 커스텀 룰)으로 만들어 기계가 잡게 하기.** 실전 프로젝트들은 이런 커스텀 규칙을 수십 개씩 쌓아 두고, 규칙의 에러 메시지에 관련 문서 경로를 넣어 AI가 스스로 교정하게 합니다.

사용자가 할 일: 같은 지적을 두 번 하게 되면 이렇게 요청하세요 — **"방금 그 실수, 다시는 못 하게 검사 규칙(ESLint 룰)으로 만들고 프로젝트 지침 문서에도 적어줘."**

💡 전체 수명주기 한눈에



3 성장 로드맵 — STEP 0 ~ 6

"오늘 처음 시작하는 사람"이 "서비스를 운영하는 사람"이 되기까지의 단계입니다. 각 단계의 완료 기준을 확인하고 다음으로 넘어가세요. 11장에 단계별 체크리스트가 있습니다.



그림 3. 비개발자 → 운영자 성장 계단

STEP 0 환경 이해 — 내가 보는 것과 AI가 보는 것

목표: 판을 이해한다

- 나는 **채팅창**과 **미리보기 화면**만 보면 됩니다. 폴더·파일·코드는 AI의 영역입니다.
- 이미지·자료 전달은 전부 **채팅창 업로드**로 합니다. AI가 "폴더에 넣어 달라"고 하면 "채팅으로 처리하자"고 하세요.
- 명령어는 외울 필요가 없습니다 — 코드 검증과 DB 변경은 AI가 알아서 실행하고, **배포는 플랫폼의 배포 페이지 버튼으로** 합니다. 당신이 기억할 것은 "무엇을 시키느냐"뿐입니다.
- 내 몫으로 남는 일: 브라우저에서 직접 테스트, 카톡 공유 미리보기 확인, 이미지·문구 제공, 그리고 **의사결정**.

STEP 1 기획 대화와 첫 화면

목표: 눈에 보이는 결과물 1개

- 아이디어를 말하면 AI가 기획 질문을 합니다. 대상 사용자·핵심 기능·분위기를 답하세요.
- 유형이 갈립니다: **랜딩페이지형**(소개·홍보 — 스크롤 연출 여부를 정함) vs **모바일 앱형**(폰 화면 폭 기준, 하단 탭 네비게이션).
- 첫 화면은 임시 데이터로 채워집니다. AI가 이렇게 안내하는 게 정상입니다: *"임시 데이터로 먼저 채울게요. 실제 데이터 저장은 나중에 관리자 패널 구성하면서 같이 해요."*
- **완료 기준:** 미리보기에서 첫 화면을 열어 보고, 고칠 점을 3개 이상 구체적으로 피드백했다.

STEP 2 화면 확장과 디자인 디렉팅

목표: 서비스의 뼈대 완성

- 화면을 하나씩 늘립니다. 기능 단위로 요청하면 AI가 데이터 정의부터 화면 조립까지 알아서 처리합니다 — 사용자는 기능 단위로 요청만 하면 됩니다.
- 디자인은 감으로 말하지 말고 **디자인 카탈로그 페이지(색·글꼴·간격 견본 모음)를 만들어 달라고 해서, 그것을 보면서** 지시하세요. 색·글꼴·간격이 토큰(공용 변수)으로 관리되면, 토큰 하나를 바꿀 때 전체에 반영됩니다.
- "AI 티 나는 디자인"(보라 그라데이션, 이모지 아이콘, 밋밋한 기본 폰트)을 경계하세요. 결과물이 어디서 본 듯하면 "더 과감하고 독창적으로"라고 요구해도 됩니다.
- **완료 기준:** 핵심 화면이 모두 존재하고(임시 데이터로), 디자인 톤이 확정됐다.

STEP 3 실데이터 전환 — DB·관리자 패널

목표: 진짜 데이터가 저장된다

- 원칙 4의 공식 관문을 엽니다. AI가 구조 점검 → 임시 데이터 정리 → 도메인별 연동 여부 확인 → 실DB 전환 순으로 진행합니다.
- 이때 **관리자 패널(/admin)**이 실데이터와 연결됩니다. AI에게 진입 경로를 꼭 확인하세요 — 주소를 모르면 있어도 못 씁니다.
- DB 구조 변경은 이후부터 반드시 마이그레이션(변경 이력 파일)으로만 합니다. "SQL로 직접 고쳐줘"라고 요청하지 마세요.
- 실데이터 전환이 끝난 뒤에도 "리팩토링 해줘"는 계속 쓸 수 있습니다 — 그때는 구조 정리와 안 쓰는 코드 청소를 하는 **유지보수 작업**이 됩니다. 기능을 여러 개 추가한 뒤 한 번씩 돌려 주는 습관이 좋습니다.
- **완료 기준:** 관리자 패널에서 데이터를 넣으면 실제 화면에 반영된다.

STEP 4 로그인·파일 업로드 — 계정과 콘텐츠

목표: 사용자가 가입하고 콘텐츠를 올린다

- "로그인 기능 넣어줘" — 인증은 직접 만들지 말고 검증된 인증 라이브러리로. **앱 사용자용과 관리자용 인증은 분리**하는 것이 표준입니다.
- "이미지 업로드 넣어줘" — 원본을 그대로 올리지 않고 **리사이즈 후 업로드**가 규약입니다(용량·비용·속도 문제).
- DB 셋업이 선행되어야 합니다(STEP 3). 순서가 어긋나면 AI가 먼저 DB부터 잡자고 할 것입니다 — 정상입니다.
- **완료 기준:** 회원가입→로그인→글쓰기/업로드→관리자 확인의 흐름이 통짜로 동작한다.

STEP 5 SEO와 배포 — 세상에 공개하기

목표: 검색되고 공유되는 서비스

- "SEO 최적화 해줘" — AI가 **공유 썸네일(가로 1200×630)**과 **파비콘(정사각 로고)**을 요청할 것입니다. 미리 준비해 두면 빠릅니다.
- 배포는 플랫폼의 배포 페이지에서 합니다. 배포 직후 **카톡에 링크를 공유해 썸네일이 잘 뜨는지** 직접 확인하세요 — 이건 사용자 몫입니다.
- **완료 기준:** 실제 주소로 접속되고, 링크 공유 시 썸네일·제목이 제대로 나온다.

STEP 6 운영자 되기

목표: 서비스를 굴리는 사람

- 결제·알림·외부 연동·법무 등 **코드 밖의 일**이 본격적으로 시작됩니다. 8장에서 상세히 다룹니다.
- 핵심 습관: 기능별 **"지금 상태 / 켜는 조건" 표**를 운영 문서로 유지하고, "아직 안 만든 것" 목록도 숨기지 말고 명시합니다.
- 이 단계에 오면 당신은 이미 요구사항 정의, 단계별 출시, 데이터 모델 감각, 장애 대응 습관을 갖춘 사람입니다 — **업계에서 이것을 하는 사람을 개발자·PM이라고 부릅니다.**

4 기술 스택, 딱 필요한 만큼만 이해하기

외울 필요 없습니다. AI가 이 도구들 얘기를 꺼낼 때 "아, 그거"라고 알아들을 정도면 충분합니다. 아래는 바이브코딩 환경에서 널리 쓰이는 대표 조합의 예시이며, 환경에 따라 구성은 달라질 수 있습니다.

이름	역할 (비유)	알아둘 것
Next.js + React	웹사이트의 골조와 내장재	화면과 서버 로직을 한 프로젝트에서 처리하는 최신 프레임워크
TypeScript	오타·착오를 미리 잡는 문법 검사기	데이터 모양(타입)을 먼저 정의하고 코드를 쓰게 함 — AI 실수 방지의 핵심 장치
Cloudflare Workers (또는 Vercel 등)	서비스가 실제로 돌아가는 서버(임대 매장)	전 세계 분산 서버에 배포됨. 플랫폼마다 기능 제약이 있어 환경별 규칙이 존재
D1 / SQLite 계열 DB + Drizzle(ORM)	데이터 창고 + 창고 관리인	ORM은 코드로 DB를 다루는 도구. 구조 변경은 반드시 마이그레이션으로
R2 / S3 계열 스토리지	이미지·파일 보관 창고	업로드 파일 저장소. 리사이즈 후 업로드가 규약
인증 라이브러리 (better-auth 등)	회원가입·로그인 기성품	인증을 직접 만들지 않고 검증된 라이브러리 사용. 앱용/관리자용 분리 운용
Tailwind CSS	디자인 규격 시스템	색·간격을 토큰으로 관리 — 디자인 카탈로그 페이지와 연동하면 디렉팅이 쉬워짐
pnpm / npm	부품(라이브러리) 조달 담당	패키지 매니저. 한 프로젝트에서는 하나만 쓰기 (혼용하면 꼬임)
ESLint	규칙 감독관	코드 규칙 자동 검사. 커스텀 규칙을 추가해 "우리 프로젝트의 금지 사항"을 기계화 (원칙 5)
Git	타임머신 / 세이브 포인트	모든 변경의 이력을 저장해 언제든 되돌릴 수 있게 함 — 9장에서 자세히

⚠ 문서는 코드보다 늦게 늙는다

프로젝트 문서에 적힌 라이브러리 버전과 실제 설치된 버전이 다른 일은 흔합니다. AI에게 버전이 중요한 작업(라이브러리 추가 등)을 시킬 때는 **"실제 설치된 버전(package.json) 기준으로 확인해서 해줘"**라고 한마디 붙이는 습관이 좋습니다.

💡 스택 선택의 단 하나의 기준

비개발자에게 최고의 스택은 "가장 유명한 것"이 아니라 **"AI가 가장 잘 아는 것"**입니다. 널리 쓰이는 표준 조합일수록 AI의 학습 데이터가 풍부해 에러가 적고 해결이 빠릅니다. 특이한 신기술을 고르면 AI도 함께 헤맵니다. AI가 제안하는 표준 구성을 받아들이는 것이 대체로 정답입니다.

5 AI에게 말하는 법 — 프롬프트 실전 가이드

같은 AI라도 시키는 사람에 따라 결과가 하늘과 땅입니다. 실전에서 검증된 표현들입니다.

단계를 여는 열쇠 문장들

이렇게 말하면	이런 일이 일어난다
"OO 서비스를 만들고 싶어. 바로 만들지 말고 기획부터 같이 잡자"	기획 대화 시작 — 확인 후 첫 화면 1개만 제작 (원칙 1·3)
"리팩토링 해줘" / "임시 데이터를 실DB로 전환하자"	구조 정리 + 실데이터 전환 관문. 도메인별 연동 선택 질문이 옴 (원칙 4)
"로그인 넣어줘" / "이미지 업로드 넣어줘"	인증·스토리지 표준 절차 실행 (DB 셋업 선행 필요)
"SEO 최적화 해줘"	썸네일·파비콘 요청 → 메타태그·사이트맵·검색 노출 일괄 세팅
"지금까지 결정된 내용을 프로젝트 지침 문서에 정리해줘"	대화가 길어져도 규칙이 유지됨 (원칙 5, 9장)

품질을 끌어올리는 요청법

- **피드백은 관찰한 사실로:** "이상해요" ❌ → "모바일에서 제목이 두 줄로 깨지고, 버튼이 엄지로 누르기엔 작아요" ✅
- **수치와 기준 제시:** "빠르게 해줘" ❌ → "첫 화면이 3초 안에 떠야 해요" ✅
- **범위 한정:** "전체 다 고쳐줘" ❌ → "일단 홈 화면의 상단 배너만" ✅
- **재발 방지 요청:** 같은 실수 2번째 → "이 실수를 다시 못 하게 검사 규칙으로 만들어줘" ✅
- **계획 먼저:** 큰 작업 전 → "바로 하지 말고, 어떻게 할 계획인지 먼저 요약해줘" ✅
- **완료 기준 명시:** "회원가입하고 로그인해서 글을 쓸 수 있으면 완료야. 직접 그 흐름을 테스트하고 결과를 보여줘" ✅

실전 대화 예시 — 첫 만남 5분이 프로젝트의 방향을 정한다

동네 베이커리 예약 서비스를 만든다고 해 봅시다. 좋은 첫 대화는 이렇게 흘러갑니다:

— 새 프로젝트, 첫 대화 —

동네 빵집 예약 서비스를 만들고 싶어. **바로 만들지 말고 기획부터 같이 잡자.**

좋아요, 몇 가지만 확인할게요. ① 손님이 쓰는 앱인가요, 사장님이 쓰는 관리 도구인가요? ② 핵심은 "미리 주문"인가요 "자리 예약"인가요? ③ 분위기는 따뜻한 동네 느낌 vs 깔끔한 프랜차이즈 느낌?

손님용이야. 케이크를 미리 주문하고 찾아가는 게 핵심. 따뜻한 동네 느낌으로.

그럼 이렇게 제안해요 — 모바일 앱형 화면으로, 홈(오늘의 빵) → 케이크 주문 → 픽업 시간 선택 → 주문 확인 흐름. 먼저 **홈 화면 1개만** 임시 데이터로 만들어 보여드릴게요. 이대로 진행할까요?

좋아, 진행해줘.

포인트: 사용자는 코드 얘기를 한 마디도 하지 않았습니다 — 대상·핵심·분위기만 답했습니다.

AI가 이렇게 말하면 잘 작동하고 있다는 신호

— 단계별로 이런 말이 나오면 정상 궤도입니다 —

💬 **첫 화면 착수 시** — "임시 데이터로 먼저 채울게요. 실제 데이터 저장은 나중에 관리자 패널 구성하면서 같이 해요."

🎨 **랜딩페이지 제작 시** — "화려한 스크롤 효과 넣을까요? (횡스크롤, 섹션 덮기, 핀 고정 등)"

🗄️ **실데이터 전환 시** — "현재 임시 데이터로 동작하는 기능들: 회원(user) / 상품(product) / 주문(order) — 전부 다 실제로 연동할까요?"

회원만 먼저. 나머지는 나중에.

📤 **SEO 작업 시** — "카톡/링크 공유할 때 뜨는 썸네일 이미지를 채팅창에 올려주세요 (가로형, 1200×630 권장)"

🔗 **DB 연동을 먼저 제안할 때** — "관리자 패널도 같이 만들어야 해요. 리팩토링하면서 DB 연동까지 진행할까요?"

※ 연동 확인에는 "네"(전체) / "회원만"(부분) / "나중에"(보류)로 답하면 됩니다.

💡 반대로, 이런 신호가 오면 개입하세요

- 첫 요청인데 질문 없이 바로 코드를 쏟아냄 → "잠깐, 기획부터 확인하자"
- 한 번에 여러 화면·기능을 통째로 만들려 함 → "첫 화면 하나만 먼저"
- 파일·폴더를 열어 보라고 안내함 → "채팅으로 보여줘 / 미리보기 주소로 알려줘"
- 에러를 "일단 우회했다"고 함 → "원인이 뭔지, 재발 방지는 어떻게 하는지 설명해줘"
- "완료했습니다"만 반복하고 확인 방법을 안 알려줌 → "내가 어디서 뭘 눌러 확인하면 되는지 알려줘"

6 구조를 알아야 하는 이유 — 4레이어와 스위치 설계

코드를 못 읽어도 됩니다. 하지만 이 구조를 알면 AI의 작업을 감독할 수 있고, 요청의 품질이 달라집니다.

4레이어 단방향 구조

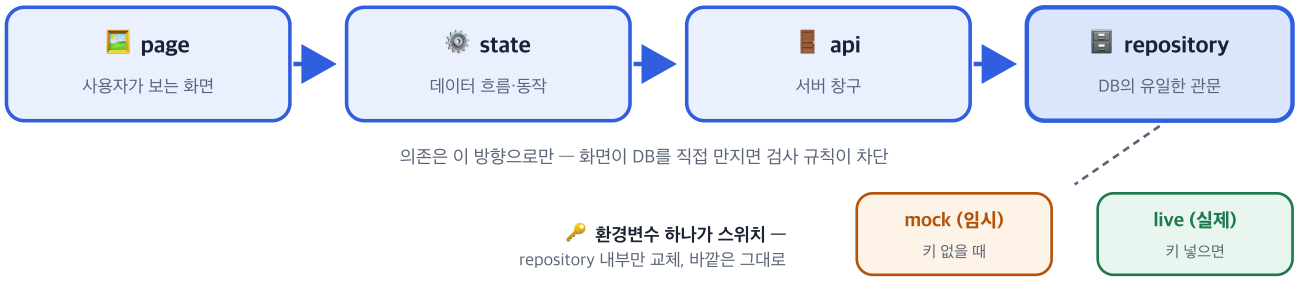


그림 4. 4레이어 단방향 구조와 mock/live 스위치 — 이 구조가 "언제든 되돌릴 수 있는 서비스"를 만든다

레이어	역할	핵심 규칙 (왜 중요한가)
page	사용자가 보는 화면	화면에서 데이터를 직접 가공(집계·필터)하지 않음 — 페이지네이션된 부분 데이터로 계산하면 **숫자가 반드시 틀리기 때문**
state	화면의 데이터 흐름과 동작	토스트·확인창·이동 같은 부수 동작을 한곳에 모음 — 동작이 흩어지면 버그 추적 불가
api	서버 쪽 창구	사용자에게 보여줄 에러는 전용 에러 타입으로 — 일반 에러는 **운영 환경에서 메시지가 가려져 사용자가 아무것도 못 봄**
repository	DB 접근의 유일한 관문	DB 구조(schema)를 가장 먼저 설계, 삭제는 soft delete 기본, 시간순 정렬되는 ID(UUID v7) 사용

화살표는 한 방향입니다. 화면이 DB를 직접 만지는 일은 커스텀 검사 규칙으로 기계적으로 차단하는 것이 정석입니다 (원칙 5).

이 구조의 진짜 보상 — "켜고 끄는 스위치"

잘 설계된 서비스의 외부 연동(DB·알림·결제 등)은 같은 패턴을 따릅니다: **"키(환경변수)가 없으면 임시 동작(mock), 키를 넣으면 실제 동작(live), 문제가 생기면 환경변수 하나로 롤백."** 이것이 가능한 이유가 바로 repository 관문 구조입니다 — 내부 구현만 mock↔live로 교체하고, 바깥(화면·상태·API)은 한 줄도 바뀌지 않습니다. "전환 시 바깥 코드가 한 줄도 안 바뀌는 것"을 완료 기준으로 삼으세요. 단, 보안이 걸린 키(인증 시크릿 등)는 반대로 **"없으면 아예 켜지지 않게"(fail-closed)** 만드는 것이 원칙입니다 — 편의는 mock으로, 보안은 차단으로.

✔ 비개발자용 요청 문장으로 번역하면

새 외부 연동을 시킬 때 이렇게 요청하세요 — **"키가 없으면 임시로 동작하고, 키를 넣으면 실제로 켜지고, 환경변수 하나로 다시 끌 수 있는 구조로 만들어줘."** 이 한 문장이 서비스의 안정성을 결정합니다. 부가 기능(알림 등)은 **"실패해도 본 기능을 막지 말고 실패 이력만 남겨줘"**도 함께.

규칙의 기계화 — 실전에서 쌓이는 커스텀 검사 규칙들

실전 프로젝트에 쌓인 커스텀 규칙들은 그대로 "AI+초보자가 자주 하는 실수 목록"입니다. 대표 사례 (7장 함정 사전과 짝을 이룹니다):

규칙(요지)	막아 주는 사고
일반 에러 던지기 금지 (전용 에러 타입 강제)	운영 환경에서 사용자에게 에러 메시지가 전혀 안 보이는 사고
api에서 DB 직접 접근 금지	관문 우회 → mock/live 스위치 구조 붕괴
배포 플랫폼 비호환 컴포넌트 금지	로컬에선 되는데 배포하면 깨지는 화면
내부 이동에 `<a>` 태그 금지	페이지 전체 새로고침으로 느려지는 문제
native `<button>` 등 금지	디자인 시스템을 벗어난 제각각 UI
파일 이름 규칙(kebab-case) 강제	대소문자 차이로 배포 환경에서만 터지는 에러

포인트: 규칙의 에러 메시지에 관련 문서 경로를 넣어 두면, 규칙에 걸린 AI가 그 문서를 읽고 스스로 고칩니다.

7 함정 사전 — 실전에서 만나는 에러와 대응법

실전 프로젝트들이 실제로 밟은 지뢰들입니다. 증상이 보이면 이 표를 AI에게 들이미세요.

개발 중 만나는 함정

증상 / 상황	원인	대응
배포하면 이미지·화면이 깨짐	배포 플랫폼과 비호환인 컴포넌트 사용 (예: 일부 플랫폼에서 Next의 `<Image>`)	플랫폼 호환 규칙 확인 — "배포 환경 기준으로 호환되는 방식으로 바꿔줘"
"로컬에서 빌드가 안 돼요"	플랫폼이 로컬 빌드를 의도적으로 차단하는 경우가 있음	정상일 수 있음. 배포는 배포 페이지에서만
개발 서버 스크립트가 특정 명령이 없다며 실패	다른 운영체제용 스크립트 실행 (리눅스용을 Mac에서 등)	"내 운영체제에 맞는 방식으로 실행해줘" — 환경 차이는 흔한 함정
"포트가 이미 사용 중" 반복	죽은 개발 서버가 포트를 잡고 있는 좀비 프로세스	프로세스 그룹째 종료 필요 — "포트 잡고 있는 좀비 프로세스 정리해줘"
화면의 통계 숫자가 실제와 다름	화면(클라이언트)에서 부분 데이터로 집계	집계는 서버에서 — "이 숫자 서버에서 계산하게 바꿔줘" (6장)
개발 DB에 데이터가 안 보임	로컬 가짜 DB를 보고 있음 (원격 DB 설정 누락)	"개발 환경이 실제 DB를 보도록 설정 확인해줘"
어제는 됐는데 오늘 갑자기 안 됨	추적 안 된 변경 (커밋 없이 쌓인 수정)	9장 — 세이브 포인트(커밋)가 있으면 "마지막으로 되던 상태와 비교해줘"가 가능

DB·인증 셋업의 함정

증상 / 상황	원인	대응
AI가 클라우드 서비스에 대화형 로그인을 하려 함	인증 방식 착오	자동화 환경의 인증은 환경변수(API 토큰)로 — "로그인 말고 토큰 방식으로"
마이그레이션 생성 실패	스키마 문법 오류, 또는 스키마 파일에 서버 전용 import가 섞임	DB 스키마 파일은 도구가 직접 읽으므로 서버 전용 표시를 넣으면 안 됨 — 흔한 함정
런타임 에러: "model not found in schema"	인증 어댑터에 스키마를 통째로 넘김	접두사 붙은 테이블은 명시적 매핑 필요 — AI에게 이 표를 보여주면 바로 잡음
로그인 가드가 뚫림/오작동	엣지 미들웨어로 인증 검사	인증 가드는 미들웨어가 아니라 **서버 컴포넌트 레이아웃**에서 — 세션 만료·폐기를 제대로 보는 위치
업로드 후 앱이 느려지고 비용 증가	원본 이미지를 그대로 업로드	리사이즈 → 업로드 → 크기 정보와 함께 저장이 규약

운영 단계의 함정

증상 / 상황	원인	대응
결제는 됐는데 주문 상태가 안 바뀜	**웹훅 URL 미등록** — 등록 안 하면 결제 해도 상태가 안 바뀐다	PG(결제사) 관리자에서 웹훅 등록 + 시크릿 설정. "웹훅이 결제의 유일한 진실원" (8장)
보낸 메일이 스팸함으로	발신 도메인 SPF/DKIM 미인증	도메인 인증 먼저 — 알림 켜기 전 필수
알림톡이 갑자기 필요해졌는데 못 씀	카카오 심사 리드타임(영업일 기준 수 일)	심사가 필요한 것들은 **개발보다 먼저 신청** (8장 리드타임 표)
배포는 됐는데 앱이 안 뜸	필수 환경변수 누락	환경변수는 로컬과 배포 환경 **양쪽에** 등록 — 한 쪽만 넣는 실수가 가장 흔함
플랫폼이 환경변수를 거부함	예약된 접두사 사용 (실제 발생 사례)	키 이름을 바꾸면 됨 — 당황할 일 아님

막혔을 때의 표준 절차 — 순서대로만 하면 대부분 풀린다



💡 에러를 만났을 때의 만능 프롬프트

"에러가 났어. ① 원인이 뭔지 비개발자도 알게 설명하고 ② 고친 다음 ③ 같은 에러가 재발하지 않게 하는 방법(규칙·문서화)까지 제안해줘." — 고치는 것에서 끝내지 않고 재발 방지까지 시키는 것이 개발자의 습관입니다.

8 실서비스 운영 — 법무·결제·알림·리드타임

코드 밖에서 서비스가 좌초하는 지점들입니다. 실서비스 운영 경험에서 정리된 항목들을 비개발자 기준으로 담았습니다.

법무 — 출시 전 필수 (P0)

> ⚠ 이 장의 법무 내용은 법률 자문이 아닙니다
> 실유저를 받기 전, 약관·개인정보처리방침 전문은 반드시 변호사 검토를 받으세요.

- **개인정보처리방침은 선택이 아니라 법적 의무**입니다(개인정보보호법 제30조). 그리고 **방침에 적은 내용과 코드가 실제로 하는 일이 일치해야** 합니다 — 수집 항목, 외부 위탁 업체, 국외 이전(해외 클라우드 인프라를 쓰면 국외 이전 고지가 필요) 모두.
- **개인은 PG(결제대행) 계약을 맺을 수 없습니다.** 결제 기능 계획이 있다면 사업자등록이 선행 조건이고, 이어서 통신판매업 신고도 필요합니다.
- 약관·방침 문서의 `[상호]`, `[사업자등록번호]` 같은 대괄호 플레이스홀더가 화면에 그대로 노출되지 않는지 출시 전 확인하세요.
- 개인정보 보호책임자(CPO) 지정 — 그 연락처 메일함은 실제로 응답 가능해야 합니다(열람·정정·삭제 요구의 법정 처리 기한은 10일입니다).
- **soft delete는 개인정보 파기가 아닙니다.** 삭제 기능을 "표시만 하는 삭제"로 만들었다면, 회원 탈퇴 시 개인정보를 '지체 없이 파기'해야 하는 법적 의무와는 별개입니다 — 탈퇴 데이터의 실제 파기 절차(수동이라도)를 운영 목록에 넣으세요.
- **정책 간 정합성:** 환불 이의제기 기간은 정산 보류 기간보다 짧게 — 돈이 나간 뒤 환불 요청이 들어오는 구조를 만들면 안 됩니다.

결제 — "웹훅이 유일한 진실원"

- 결제 완료 판정은 **결제사가 서버로 보내는 웹훅**으로만 합니다. 사용자가 보는 "결제 완료 화면"은 상태를 바꾸지 않습니다(위조 가능하기 때문). 서버는 금액·상태를 결제사 API로 재확인합니다.
- 웹훅 서명 검증(위조 차단), 중복 웹훅 무시(멱등 처리)가 기본기입니다 — AI에게 "결제는 웹훅 기준, 서명 검증과 멱등 처리 포함해서"라고 요청하세요.
- 수수료 계산은 **차감식**으로: 수수료 = 반올림(금액×요율), 지급액 = 금액−수수료 — 각각 따로 계산하면 잔돈이 유실됩니다. 요율은 정산 시점 값을 스냅샷으로 저장.
- 로컬 개발 환경에서는 웹훅을 받을 수 없습니다 — 결제 테스트는 배포된 주소에서.
- **정산금 지급 모델이라면 세금이 따라옵니다.** 개인 전문가에게 지급하는 정산액에는 원천징수(사업소득 3.3%) 의무가 걸릴 수 있습니다. 지급 기능은 "보내는 코드"보다 "세무사 확인"이 먼저입니다.

알림 — 폴백 사다리와 심사 리드타임

- 표준 구조: **알림톡 → 이메일 → 콘솔(기록만)** 순서의 폴백 사다리. 앞이 실패하면 다음으로 넘어가고, **전부 실패해도 본 기능은 멈추지 않고** 이력만 남깁니다.
- 카카오 알림톡은 템플릿 심사에 영업일 기준 수일이 걸립니다. **변수 하나만 바꿔도 재심사**이므로 문구를 확정한 뒤 신청하고, 긴 내용은 이메일 본문에만 넣습니다.
- 심사 반려의 대표 사유는 "광고성" 판정 — 템플릿 문구는 정보성으로 작성합니다.
- **자동 알림에 유료 제안을 섞을 때의 원칙:** 첫 알림에는 스스로 해결하는 방법만 담고, 유료 도움은 재발했을 때 "선택지 중 하나"로만 제시합니다 — 막힌 사람에게 바로 결제를 들이밀면 도구가 아니라 영업으로 읽히기 때문입니다.

자동화 기능은 "섀도 모드"부터

자동 발송·자동 판정 기능은 **일정 기간 감지만 하고 아무 행동도 안 하는 섀도 모드**로 출시해, 쌓인 데이터를 보고 기준값을 조정한 뒤 활성화하는 것이 안전합니다: **관찰 → 튜닝 → 활성화.** 오탐 방어 원칙도 함께 기억하세요 — "도구가 아픈 것(서버 오류)과 사용자가 막힌 것을 구분하라", "단일 신호로 발동하지 마라", "같은 사람에게 반복 발송하지 마라".

외부팀·타 서비스와 협업할 때 — 요청서 1장의 힘

다른 팀·서비스에 연동을 요청할 때의 검증된 구조: ① **왜 필요한가를 비개발자 언어로**(예: "지금은 모든 방문자가 같은 테스트 유저 — 이대로 열면 남의 데이터가 보인다") ② **최소 요구와 선택 요구를 분리**("이것 하나만 있어도 성립") ③ **상대가 할 일을 복붙 가능한 예시로.** 계약 규칙도 문서에 명문화하세요: 필드 추가는 자유, 제거·변경은 금지, 큰 변경은 새 버전(V2)으로.

리드타임 표 — 개발보다 먼저 신청할 것들

항목	통상 소요	비고
카카오 알림톡 템플릿 심사	영업일 기준 수일	변수 하나 변경도 재심사 — 문구 확정 후 신청
PG(결제) 가맹 심사	약 1~2주	사업자등록·통신판매업 신고 선행
앱스토어 심사 (앱 출시 시)	수일~수주	개발 완료를 기다리지 말고 계정·심사 준비 먼저
메일 도메인 인증 (SPF/DKIM)	수시간~1일	알림 기능 켜기 전 필수

출시 D-30 역산 캘린더 — 리드타임을 일정으로 번역하면

위 항목들을 실제 달력에 올리면 이렇게 됩니다. 결제·알림이 없는 서비스는 해당 항목을 건너뛰면 됩니다.

D-30 — **사업자등록 신청 · 도메인 구입**
결제 계획이 있다면 여기가 진짜 출발선 — 코드보다 서류가 먼저

D-21 — **통신판매업 신고 · PG(결제) 가맹 심사 신청**
가맹 심사 1~2주 — 개발과 병행

D-14 — **알림톡 템플릿 문구 확정 → 심사 신청 · 약관/방침 초안 → 변호사 검토 의뢰**
변수 하나만 바꿔도 재심사 — 문구를 먼저 확정

D-7 — **메일 도메인 인증(SPF/DKIM) · 썸네일/파비콘 준비 · SEO 셋업 · 보안 점검**
"노출된 키, 인증 없는 관리자 기능, 남의 데이터 보이는 API 점검해줘" (9장)

D-1 — **배포 → 첫 관리자 계정 생성 → 웹훅 등록 → 실결제 리허설**
결제 테스트는 반드시 배포된 주소에서 (로컬은 웹훅 수신 불가)

D-DAY — **공개🎉 카톡 공유해 썸네일 확인 · 첫 사용자 흐름 관찰**
공개는 끝이 아니라 관찰의 시작 — 운영 문서에 "지금 상태" 기록

⚠ 배포 직후 가장 먼저 할 일 — 관리자 계정

권장 설계: 관리자 가입 페이지는 **관리자가 0명일 때만 열립니다.** 배포 직후 즉시 첫 관리자 계정을 만드세요 — 그 순간 가입 문이 자동으로 잠깁니다. 이런 "처음 한 번만 열리는 문" 구조 자체가 AI에게 요청할 가치가 있는 안전장치입니다.

9 AI와 오래 일하는 법 — 검증·보안·백업·비용·멘탈

하루 이틀이 아니라 몇 달을 AI와 일하려면 필요한 습관들입니다. 이 장의 습관이 "운 좋게 완성한 사람"과 "계속 만들 수 있는 사람"을 가릅니다.

검증 — "된다"는 말과 "되는 것"은 다르다

AI의 가장 위험한 습성은 **확신에 찬 말투로 틀리는 것**입니다. "완료했습니다"는 보고이지 증거가 아닙니다. 모든 작업은 3단으로 검증하세요:

단계	방법	잡아내는 것
① **눈으로**	미리보기 화면을 직접 연다	화면이 아예 안 뜨거나 깨진 경우
② **손으로**	실제 사용자처럼 흐름 전체를 눌러본다 (가입→로그인→핵심 기능)	버튼이 안 눌리거나, 흐름 중간이 끊긴 경우
③ **데이터로**	관리자 패널에서 방금 한 행동이 기록됐는지 확인한다	화면만 그럴듯하고 실제 저장은 안 되는 경우

- AI에게도 검증을 시키세요: **"직접 그 흐름을 테스트하고, 확인한 결과(스크린샷·로그)를 보여줘."**
- AI가 사실을 지어내는 경우(환각)도 있습니다. 요금·법규·외부 서비스 정책처럼 **돈과 법이 걸린 정보는 반드시 공식 사이트에서 재확인**하세요.
- 모르는 것을 모른다고 말하게 하세요: "확실하지 않으면 추측이라고 표시해줘."

보안 — 비개발자가 가장 크게 다치는 곳

- **비밀키(API 키·시크릿)는 절대 코드나 채팅에 그대로 붙여넣지 마세요.** 코드에 들어간 키는 저장소 이력에 영원히 남고, 유출되면 요금 폭탄·계정 탈취로 이어집니다. 키는 항상 환경변수(.env)로 — 그리고 .env 파일은 저장소에 올라가지 않게(".env가 gitignore에 있는지 확인해줘").
- 키가 한 번이라도 채팅·코드에 노출됐다면 **지우는 것으로 끝이 아니라 재발급**이 원칙입니다.
- 화면 캡처를 공유할 때 관리자 주소·이메일·키가 찍혀 있지 않은지 확인하세요.
- 출시 전 AI에게 한 번 시키세요: **"보안 점검해줘 — 노출된 키, 인증 없이 접근 가능한 관리자 기능, 남의 데이터가 보이는 API가 있는지."**

세이브 포인트 — 버전 관리(Git)의 최소 상식

Git은 게임의 세이브 포인트입니다. 커밋(저장)해 두면 언제든 그 시점으로 돌아갈 수 있습니다. 명령어는 몰라도 됩니다 — **습관만 가지세요:**

- **되는 상태가 될 때마다:** "지금 상태 커밋해줘" — 특히 큰 작업(리팩토링, DB 전환) 직전에는 반드시.
- **망가졌을 때:** "마지막 커밋으로 되돌려줘" 또는 "마지막으로 되던 상태와 지금의 차이를 보여줘".
- 커밋 없이 몇 시간을 달리는 것은 세이브 없이 보스전에 들어가는 것과 같습니다.
- DB의 실데이터는 커밋으로 보호되지 않습니다 — 운영 시작 후에는 **DB 백업**을 따로 챙기세요("DB 백업은 어떻게 되고 있는지 알려줘").

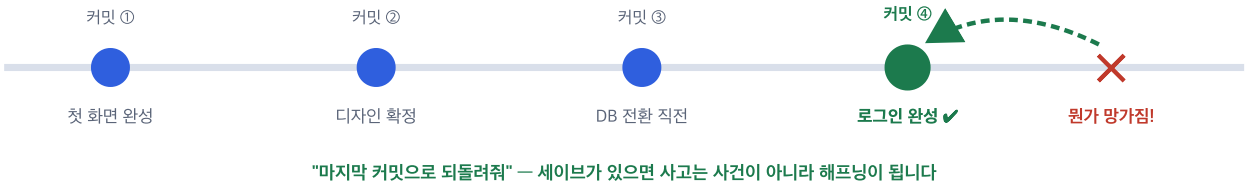


그림 5. 세이브 포인트(커밋)의 힘 — 되는 상태마다 저장해 두면 언제든 돌아올 수 있다

대화(컨텍스트) 관리 — AI의 기억력에는 한계가 있다

- 대화가 길어지면 AI는 앞부분을 잊거나 뭉개서 기억합니다. **품질이 떨어진다고 느껴지면 대화를 새로 시작**하는 것이 종종 가장 빠른 해결책입니다.
- 새 대화가 이어받을 수 있도록, 결정 사항은 대화가 아니라 **문서에 남기세요:** "지금까지 결정된 규칙과 진행 상황을 프로젝트 지침 문서에 정리해줘." — AI가 매번 읽는 지침 문서(CLAUDE.md 등)가 프로젝트의 헌법이 됩니다.
- 작업 단위로 대화를 끊으세요: 한 대화 = 한 기능. "지난 대화에서 하던 것 이어서"보다 "지침 문서 읽고 OO 기능 시작하자"가 안정적입니다.

비용 감각 — 놀라지 않기 위한 최소한

- 비용은 크게 셋입니다: **AI 사용료**(구독·토큰), **인프라**(서버·DB·스토리지 — 초기엔 무료 티어로 대부분 충분), **외부 서비스**(결제 수수료, 알림 발송비, 도메인 연 1~2만원).
- 과금 사고의 단골: 원본 대용량 이미지 업로드(스토리지·전송량), 무한 반복 호출 버그, 노출된 API 키 도용. — 앞의 규약(리사이즈 업로드)과 보안 습관이 곧 비용 방어입니다.
- 클라우드 콘솔의 **예산 알림(billing alert)**을 켜 두세요. "월 1만원 넘으면 메일" 한 줄이 폭탄을 막습니다.

멘탈과 성장 — 막히는 것이 정상입니다

- **막힘은 실력 부족의 증거가 아니라 개발의 기본 상태입니다.** 전문 개발자도 하루의 상당 부분을 막힌 채 보냅니다. 다만 그들은 "막혔을 때의 절차"를 가지고 있을 뿐입니다: 에러 메시지를 읽는다 → 재현 조건을 좁힌다 → 마지막으로 되던 상태와 비교한다 → 그래도 안 되면 도움을 청한다.
- 도움을 청할 때는 **"안 돼요"가 아니라 재현 절차로:** "① 이걸 눌렀더니 ② 이런 화면에서 ③ 이 메시지가 떴다 (스크린샷)" — 이렇게 묻는 사람이 가장 빨리 답을 얻습니다.
- 하루에 용어 하나면 충분합니다. 오늘 AI가 쓴 단어 중 하나를 골라 "이거 비유로 설명해줘"라고 물어보세요. 3개월이면 대화의 해상도가 완전히 달라집니다.
- **완벽주의를 경계하세요.** 기능을 더 붙이고 싶은 유혹은 출시를 미루는 가장 달콤한 핑계입니다. "부끄러운 첫 버전을 빨리 내고, 사용자의 반응으로 다음을 정한다" — 이것이 MVP 정신입니다.

✔ 이 장을 한 문장으로

믿지 말고 확인하라, 키는 숨기고, 세이브는 자주, 결정은 문서에, 지출엔 알람을, 그리고 막혀도 계속.

10 위험 구간 지도 — 전문지식이 꼭 필요한 7가지 순간

AI는 "돌아가는 것"을 잘 만듭니다. 문제는 **"돌아가는 것처럼 보이는데 틀린 것"**이 만들어지는 영역입니다. 전문 개발자들이 사고를 겪으며 경력으로 배우는 부분 — 비개발자는 이 장의 요청 문장과 테스트법으로 대신하세요.

① 권한 — "내 것만 보이는가"는 눈에 보이지 않는다

바이브코딩 서비스의 단골 보안 사고: 화면에는 내 글만 보이는데, **주소(URL)의 번호만 바꾸면 남의 글·주문·개인정보가 열리는** 경우입니다. 비유하면 "통장 번호만 대면 아무에게나 잔액을 알려주는 은행"입니다. 화면에 버튼이 없는 것은 안내일 뿐, 진짜 잠금은 서버에 있어야 합니다 — 그리고 이건 **화면을 아무리 눌러 봐도 발견할 수 없습니다.**

> ✔ 비개발자의 방어법
> **요청:** "모든 API에서 '이 데이터가 요청한 사람의 것인지'를 서버에서 확인하는지 전수 점검해줘. 화면 기준이 아니라 서버 기준으로."
> **직접 테스트:** 계정을 2개 만들어, A 계정으로 만든 글의 주소를 B 계정으로 로그인한 브라우저(또는 시크릿 창)에서 열어 보세요. 열람뿐 아니라 **수정·삭제도 시도**해야 합니다. 관리자 페이지 주소도 일반 계정으로 열어 보세요.

② 돈 계산 — 동시성과 "전부 아니면 전무"

실제로 일어나는 사고 세 가지:

- **마지막 재고 1개를 두 명이 동시에 결제 → 둘 다 성공.** 컴퓨터는 요청을 동시에 처리하므로 "확인하고 나서 차감"의 틈새에 두 번째 요청이 끼어듭니다.
- **결제 버튼 연타 → 포인트 이중 사용 / 주문 2건 생성.** 사용자는 반드시 연타합니다.
- **포인트는 차감됐는데 주문 생성은 실패 → 반쪽 성공.** 관련 변경은 트랜잭션(전부 성공 아니면 전부 취소)으로 묶여야 합니다.

> ✔ 비개발자의 방어법
> **요청:** "재고 차감·포인트 사용·잔액 변경은 동시에 두 요청이 와도 안전하게 처리하고, 연관된 변경(차감+주문 생성)은 트랜잭션으로 묶어서 반쪽 성공이 없게 해줘. 같은 요청이 중복으로 오면 한 번만 처리되게(멱등) 해줘."
> **직접 테스트:** 구매·신청 버튼을 빠르게 연타해 보세요. 주문이 2건 생기거나 포인트가 두 번 빠지면 사고입니다.

③ 데이터 설계 — 지금 대충 잡으면 나중에 백배로 비싸다

화면은 도배지라 언제든 바꿀 수 있지만, **DB 구조(스키마)는 건물의 기초**라 실데이터가 쌓인 뒤 바꾸는 것은 이사 수준의 공사입니다. 대표 사고: 주문이 상품 가격을 "참조"만 하게 만들면, 사장님이 가격을 수정하는 순간 **과거 주문의 금액까지 바뀝니다** — 회계상 영수증 위조가 되는 셈입니다.

> ✔ 비개발자의 방어법
> **요청:** "주문·계약·정산처럼 '시점'이 중요한 데이터에는 그 시점의 가격·상품명·조건을 복사(스냅샷)해서 저장해줘. 상태(status) 값은 자유 문자열 말고 허용 값 목록으로 제한해줘."
> **타이밍:** 실DB 전환(원칙 4) 직전에 한 번 — "이 스키마로 1년 뒤에도 문제없을지, 특히 시점 데이터·삭제 정책·상태 값 관점에서 검토해줘."

④ 성능 — 유저 100명까지는 뭘 해도 빠르다

데이터 10건일 때는 어떤 코드도 빠릅니다. 문제는 1만 건부터입니다. 대표 원인: 목록 하나를 그리는 데 DB를 100번 호출하는 구조(N+1), 자주 검색하는 컬럼에 색인(인덱스)이 없는 것. 출시 후 "갑자기 느려졌어요"의 정체가 대부분 이것입니다.

✔ 비개발자의 방어법

요청: "가짜 데이터 1만 건을 넣고 주요 화면의 속도를 테스트해줘. 목록 화면이 DB를 몇 번 호출하는지, 자주 검색·정렬되는 컬럼에 인덱스가 있는지도 점검해줘."

단, 과유불급: 출시 전엔 이 테스트 한 번이면 충분합니다. 유저도 없는데 성능 최적화에 시간을 쓰는 것은 대표적인 출시 지연 사유입니다.

⑤ 환경 분리 — 운영 DB에서 실험하지 마라

"잠깐 테스트해 본다"가 실고객 데이터 삭제로 이어지는 사고는 전문 개발자도 당합니다. 개발용과 운영용 환경을 분리하고, 운영 데이터 위에서의 실험은 금지가 원칙입니다. 그리고 **백업이 "있다"는 것과 "복구가 된다"는 것은 다른 문제입니다** — 복구를 한 번도 안 해 본 백업은 부적입니다.

✔ 비개발자의 방어법

요청: "대량 삭제·구조 변경 같은 위험한 작업은 실행 전에 영향 범위(몇 건이 바뀌는지)를 먼저 보여주고, 운영 DB 백업 후 진행해줘."

리허설: 운영 시작 후 한 번은 실제로 백업에서 복구를 테스트해 보세요 — "어제 백업으로 복구하는 절차를 처음부터 끝까지 같이 해 보자."

⑥ 소유권 — 계정과 자산은 전부 내 명의로

도메인, 클라우드 계정, PG 계약, 앱스토어 계정 — 이것들이 서비스의 등기부등본입니다. 흔한 사고: 지인·외주 명의로 등록한 도메인 → 관계가 틀어지면 **서비스 주소 자체를 잃습니다.** 코드보다 계정을 잃는 것이 훨씬 치명적입니다.

- 도메인·클라우드·PG·앱스토어 계정은 **전부 내 명의, 내 이메일**로 — 그리고 2단계 인증 필수, 복구 이메일·전화번호 최신화.
- 플랫폼 종속 대비: "코드와 DB를 통째로 내보내려면(export) 어떻게 하는지 알려줘"를 미리 확인해 두세요. 떠날 수 있는 사람이 좋은 조건으로 머뭅니다.

⑦ 전문가 호출 기준 — AI로 될 일과 안 될 일

AI만으로 끝까지 가려는 것도, 모든 걸 전문가에게 맡기려는 것도 답이 아닙니다. 경계선은 이렇습니다:

영역	AI로 충분한 일	사람 전문가가 필요한 일
개발	화면·기능 제작, 에러 수정, 구조 점검 — 이 가이드의 거의 전부	—
보안	일상 점검 (권한 테스트, 키 관리, 이 장의 ①)	결제·개인정보를 대량 보유하는 서비스의 출시 전 보안 점검(모의해킹)
법무	약관·방침 초안 작성, 개념 이해	약관·방침 최종본 검토, 분쟁·분쟁 예고 발생 시 (8장)
세무	개념 이해, 자료 정리	원천징수·부가세 신고, 정산 자동화 전 설계 검토 (8장)
사고 대응	일상 장애의 원인 분석·복구	개인정보 유출 사고 — 인지 후 **72시간 내 신고·통지 의무**가 있으므로 즉시 변호사와 함께 대응

💡 출시 전, 이 장 전체를 한 번에 시키는 통합 점검 프롬프트

"출시 전 위험 구간 점검을 해줘: ① 모든 API의 소유권 검증(남의 데이터 접근 가능 여부) ② 돈·재고·포인트 로직의 동시성과 트랜잭션 ③ 시점 데이터의 스냅샷 저장 여부 ④ 가짜 데이터 1만 건 성능 테스트 ⑤ 위험 작업의 백업 절차 — 각 항목의 결과를 표로 보고해줘."

11 종합 체크리스트

단계별로 인쇄해 두고 체크하며 진행하세요. "왜"가 궁금하면 괄호의 장 번호로 돌아가면 됩니다.

A. 시작 전 (STEP 0~1)

- [ ] 만들려는 서비스를 한 문장으로 설명할 수 있다 (누가 · 무엇을 · 왜)
- [ ] AI의 기획 질문에 답했고, "이대로 진행할까요?" 확인을 거쳤다
  원칙 1 — 기획 대화 없이 시작한 프로젝트가 가장 많이 갈아엎어진다
- [ ] 첫 화면 1개만 임시 데이터로 만들었다
  원칙 2·3 — Screen First, 한 번에 한 화면
- [ ] 미리보기를 직접 열어 보고 구체적 피드백을 3개 이상 줬다

B. 화면 확장기 (STEP 2)

- [ ] 요청을 화면/기능 단위로 쪼개서 하고 있다
- [ ] 디자인 지시는 디자인 카탈로그 페이지를 보면서 한다 (3장 STEP 2)
- [ ] 모바일 화면에서도 직접 확인했다 (폰으로 접속)
- [ ] 같은 실수 지적이 2번 반복되면 "검사 규칙으로 만들어줘"라고 요청했다
  원칙 5 — 말은 잊히고 규칙은 남는다
- [ ] 되는 상태가 될 때마다 "커밋해줘"로 세이브 포인트를 만들고 있다 (9장)

C. 실데이터 전환기 (STEP 3~4)

- [ ] 전환 직전에 커밋(세이브 포인트)을 만들었다 (9장)
- [ ] 공식 관문으로 전환했다 (임의로 DB부터 만들지 않았다)
  원칙 4
- [ ] 연동할 도메인을 선택했다 — 전부일 필요 없음 ("회원만 먼저"도 답)
- [ ] 관리자 패널 주소를 안내받았고, 접속·로그인해 봤다
- [ ] 관리자 패널에서 데이터를 넣으면 실제 화면에 반영되는 것을 확인했다
- [ ] DB 구조 변경은 마이그레이션으로만 한다 (SQL 직접 수정 요청 금지)
  7장 — 이력 없는 DB 변경은 복구 불가능한 사고의 씨앗
- [ ] 회원가입→로그인→핵심 기능→관리자 확인, 전체 흐름을 통짜로 테스트했다
  9장 3단 검증 — 눈으로 · 손으로 · 데이터로

D. 공개 준비 (STEP 5)

- [ ] 공유 썸네일(1200×630)과 파비콘(정사각)을 준비해 SEO 셋업을 마쳤다
- [ ] 환경변수를 로컬과 배포 환경 양쪽에 등록했다
  7장 — 한쪽만 넣는 실수가 가장 흔한 배포 사고
- [ ] 비밀키가 코드·채팅에 노출된 적 없는지 점검했고, 노출됐던 키는 재발급했다 (9장)
- [ ] 출시 전 보안 점검을 시켰다 — 인증 없는 관리자 기능, 남의 데이터가 보이는 API (9장)
- [ ] 배포 후 실제 주소로 접속 확인했다

- [ ] 카톡에 링크를 공유해 썸네일·제목이 제대로 뜨는지 확인했다
  이건 AI가 못 하는, 사용자 몫의 테스트
- [ ] 배포 직후 첫 관리자 계정을 만들었다 (가입 문이 잠기는지 확인) (8장)

E. 운영 개시 — 법무·결제·알림 (STEP 6)

- [ ] 개인정보처리방침·이용약관이 있고, 대괄호 플레이스홀더가 화면에 남아 있지 않다
- [ ] 방침에 적힌 수집 항목·위탁 업체·국외 이전 내용이 실제 코드 동작과 일치한다 (8장)
- [ ] 약관·방침 전문에 대해 변호사 검토를 받았다 (8장)
- [ ] 회원가입 화면에 약관·개인정보처리방침 동의 체크박스와 문서 링크가 있다 (동의 시각을 저장할지도 결정했다)
- [ ] 회원 탈퇴 시 개인정보 파기 절차를 운영 목록에 넣었다 (soft delete와 별개) (8장)
- [ ] (결제 시) 사업자등록 → 통신판매업 신고 → PG 가맹 심사를 개발과 병행해 미리 진행했다
- [ ] (결제 시) 웹훅 URL과 시크릿을 결제사 관리자에 등록했다
  등록 안 하면 결제해도 상태가 안 바뀐다
- [ ] (결제 시) 결제 테스트는 배포된 주소에서 했다 (로컬은 웹훅 수신 불가)
- [ ] (알림 시) 발신 도메인 SPF/DKIM 인증을 마쳤다 (스팸함 방지)
- [ ] (알림 시) 알림톡 템플릿은 문구 확정 후 신청했고, 심사 리드타임을 일정에 반영했다
- [ ] 환불 이의제기 기간 < 정산 보류 기간으로 정책 정합성을 맞췄다
- [ ] 알림 등 부가 기능이 실패해도 본 기능이 멈추지 않는 구조인지 확인했다

F. 운영 습관 — 지속 가능한 서비스

- [ ] 기능별 "지금 상태 / 켜는 조건" 표를 운영 문서로 유지한다
  미래의 나와 협업자를 위한 인수인계
- [ ] "아직 안 만든 것" 목록을 문서에 명시한다 (숨기면 잊는다)
- [ ] 새 외부 연동은 항상 "키 없으면 임시 동작, env 하나로 롤백" 구조로 요청한다 (6장)
- [ ] 자동화 기능은 섀도 모드(관찰만) → 데이터 확인 → 활성화 순서로 켠다 (8장)
- [ ] 운영 DB의 백업이 어떻게 되고 있는지 확인했다 (9장)
- [ ] 클라우드 예산 알림(billing alert)을 켜 뒀다 (9장)
- [ ] 결정 사항은 대화가 아니라 프로젝트 지침 문서에 쌓는다 (9장)
- [ ] 외부팀 요청은 "왜 + 최소 요구 + 상대가 할 일 예시"를 담은 요청서 1장으로 한다 (8장)
- [ ] 심사·계약 등 리드타임이 있는 일은 개발보다 먼저 신청한다 (8장 표)

G. 위험 구간 점검 — 출시 전 반드시 (10장)

- [ ] 계정 2개로 남의 데이터 열람·수정·삭제를 시도해 봤다 (전부 막혀야 정상)
  10장 ① — 화면을 눌러서는 절대 발견할 수 없는 사고
- [ ] 구매·신청·포인트 버튼을 연타해 봤다 (중복 처리되면 사고)
  10장 ② — 사용자는 반드시 연타한다
- [ ] 주문·계약 등 시점 데이터에 가격·조건 스냅샷이 저장되는지 확인했다
  10장 ③ — 가격 수정이 과거 기록을 바꾸면 안 된다

- [ ] 가짜 데이터 1만 건을 넣고 주요 화면 속도를 테스트했다 (10장 ④)
- [ ] 백업에서 실제로 복구하는 리허설을 한 번 해봤다
  10장 ⑤ — 복구 안 해 본 백업은 부적이다
- [ ] 도메인·클라우드·PG·앱스토어 계정이 전부 내 명의이고, 2단계 인증이 켜져 있다 (10장 ⑥)
- [ ] 전문가 검토가 필요한 항목(보안·법무 최종본·세무)을 확인하고 일정에 넣었다 (10장 ⑦)
- [ ] 통합 점검 프롬프트로 5개 항목 결과를 표로 보고받았다 (10장)

부록 비개발자를 위한 용어 사전

AI와의 대화에 등장하는 단어들. 뜻만 알아도 대화의 해상도가 올라갑니다.

용어	뜻
목데이터 (mock)	진짜처럼 보이게 채워 넣은 임시 데이터. 화면 확인용이며, 나중에 실데이터로 교체됨
배포 (deploy)	내 컴퓨터에서만 보이던 것을 인터넷 주소로 세상에 공개하는 일
환경변수 (env)	비밀 열쇠·설정값을 코드 밖에 보관하는 방식. "키를 넣으면 켜진다"의 그 키
마이그레이션	DB 구조 변경의 이력 파일. 언제 무엇이 바뀌었는지 기록되어 재현·복구가 가능해짐
스키마 (schema)	DB의 설계도 — 어떤 표(테이블)에 어떤 칸(컬럼)이 있는지의 정의
API / Server Action	화면이 서버에 일을 시키는 창구
웹훅 (webhook)	외부 서비스(결제사 등)가 "일이 끝났다"고 내 서버에 보내 주는 자동 연락
세션 (session)	"이 사람이 로그인한 그 사람이 맞다"는 서버의 기억
미들웨어	모든 요청이 지나가는 관문 코드. 단, 인증 검사는 여기 말고 레이아웃에서 (7장)
린트 (lint)	코드 규칙 자동 검사. 커스텀 규칙을 추가해 "우리 프로젝트의 금지 사항"을 기계화할 수 있음
리팩토링	기능은 그대로 두고 내부 구조를 정리하는 일. 실데이터 전환의 관문 명령으로도 쓰임
커밋 (commit)	Git의 세이브 포인트. "지금 상태 커밋해줘" = 언제든 돌아올 수 있는 저장
Git / 저장소	코드 변경 이력을 전부 기록하는 타임머신. 되돌리기·비교의 근거
토큰 (디자인)	색·간격·글꼴을 이름 붙여 관리하는 공용 변수. 토큰 하나를 바꾸면 전체에 반영
폴백 (fallback)	1안이 실패하면 2안으로 자동 전환하는 구조 (알림톡→이메일→기록)
멱등 (idempotent)	같은 요청이 여러 번 와도 결과가 한 번과 같음 — 중복 웹훅·중복 클릭 방어의 원리
섀도 모드	기능을 실제로 발동하지 않고 "발동했다면"의 기록만 쌓는 관찰 모드
soft delete	데이터를 진짜 지우지 않고 "지워짐" 표시만 하는 삭제 — 복구 가능성 확보 (단, 개인정보 파기 의무와는 별개 — 8장)
환각 (hallucination)	AI가 사실이 아닌 것을 확신에 차서 말하는 현상 — 돈·법 관련 정보는 반드시 재확인 (9장)
컨텍스트	AI가 한 대화에서 기억할 수 있는 범위. 길어지면 앞부분을 잊음 — 결정은 문서로 (9장)
MVP	Minimum Viable Product — 핵심 기능만 담은 최소 출시 버전. 완벽 대신 빠른 검증
트랜잭션	여러 변경을 하나로 묶어 "전부 성공 아니면 전부 취소"로 처리하는 것 — 반쪽 성공 방지 (10장)
동시성 (race)	두 요청이 동시에 처리될 때 생기는 충돌 — 마지막 재고를 두 명이 사는 사고의 원인 (10장)
스냅샷	그 시점의 값(가격·조건)을 복사해 저장하는 것 — 원본이 바뀌어도 기록은 보존 (10장)
인덱스 (색인)	DB의 찾아보기 — 없으면 데이터가 쌓일수록 검색이 급격히 느려짐 (10장)
라우트 그룹	주소(URL)에는 드러나지 않게 화면들을 묶는 폴더 규칙 — 괄호 이름의 폴더, 예: (app)
바인딩 (binding)	배포 환경에서 내 앱과 DB·저장소를 연결해 주는 설정
UUID v7	시간순 정렬이 되는 고유 ID 형식
PG (결제대행)	카드사와 서비스 사이에서 결제를 대행하는 회사(포트원 등). 계약에 사업자등록 필요

✔ 마지막으로 — "개발자가 된다"의 실제 의미

이 가이드를 따라 서비스 하나를 출시하고 운영해 본 사람은, 요구사항을 정의하고 · 단계를 설계하고 · 데이터 구조를 감각하고 · 장애에 대응하고 · 재발을 방지하는 사람이 됩니다. 코드를 손으로 치는 것은 이미 AI의 일입니다. **순서를 알고, 규칙을 만들고, 결정을 내리는 것 — 그것이 이 시대의 개발자입니다.**