돈이 지나가는 소프트웨어

구독, 결제, 환불의 기술

ThakiCloud (Hyojung Han)

돈이 지나가는 소프트웨어

구독, 결제, 환불의 기술

ThakiCloud (Hyojung Han)

Contents

1 돈이 흐르는 자리가 다른 이유 1

2 구독 청구 설계: 날짜, 상태, 실패한 카드 8

3 결제 레일과 환불: 돈이 오가는 길 15

4 웹훅, 대조, 그리고 세금 경계 22

1 돈이 흐르는 자리가 다른 이유

서비스는 잘 돌아가고 있다. 그런데 고객의 카드는 지난달에 만료됐다. 시스템은 청구일에 결제를 시도했고 실패했다. 이틀 뒤에 다시 시도했고 또 실패했다. 그리고 아무 일도 일어나지 않았다. 상태 변화도, 메일도, 로그도 없다. 고객은 한 달 내내 서비스를 계속 썼다. 그 달의 매출은 존재하지 않는다. 관리자 화면을 열어봐도 그 구독은 여전히 활성으로 보인다.

이런 실패는 에러를 남기지 않는다. 크래시도, 500도 없다. 돈 시스템의 최악의 실패 방식은 결제 실패 자체다. 시스템이 그 사실을 모를 때 진짜 위험이 시작된다.

1.1 CRUD와 다른 세 가지

1인 개발자가 익숙한 코드에는 이런 성질이 있다. 모든 것이 한 프로세스 안에 있다는 것이다. 데이터베이스도, 파일도, 상태도 거기 있다. 틀리면 롤백하면 된다. 롤백한 뒤 세상은 다시 이전으로 돌아간다.

결제는 그렇지 않다. 세 가지가 다르다.

첫째, 상대가 내 프로세스 밖에 있다. 결제 요청은 네트워크를 지나 결제 서비스 제공사(이하 PSP)로 가고 PSP는 카드사나 은행으로 가고 거기서 답이 돌아온다. 이 각각은 별개의 회사고 별개의 장애 방식을 갖고, 별개의 시계를 갖는다. 내 서버가 10분간 다운되는 동안에도 PSP는 자기 장애를 겪고 있고, 은행의 정산은 자기 스케줄대로 돈다. 한 트랜잭션으로 묶어서 끝낼 수 없다.

둘째, 시간이 시스템에 들어온다. 구독은 청구 주기가 있고 결제는 정산 지연이 있고 환불은 카드에 도착하는 데 며칠이 걸린다. API가 반환된 순간이 끝이 아니다. API 응답은 접수됐다는 뜻이고 실제 사실이 나중에

온다. 그것도 이벤트라는 형태로.

셋째, 효과를 취소할 수 없다. 데이터베이스의 행은 지우면 사라진다. 돈은 움직인 뒤 되돌려 보낼 수 있을 뿐이다. 환불은 새로운 이동이며, 그 이동에는 고유한 실패 방식이 있다. 그래서 돈 시스템에서 되돌리는 것은 항상 이벤트다. 취소가 아니다.

이 셋이 합치면 이런 말이 나온다. 돈 시스템에서는 정보가 완비되고 시간이 순서대로 흐른다는 가정은 성립하지 않는다. 정보가 유실되고 중복되고 순서 없이 도착하는 세계를 전제로 설계해야 한다. 이 책의 모든 내용은 그 세계에서 미치지 않기 위한 방법이다.

1.2 돈의 세 다리

구독 사업에서 돈이 움직이는 곳은 세 곳이다. 청구, 수금, 환불.

다리	주요 이벤트	흔한 실패
청구	금액 계산, 청구서 발행	중간 정산 오류, 주기 누락
수금	돈 이동, 성공 확인	중복 청구, 웹훅 유실
환불	돈 되돌리기	환불 후 상태 미갱신

세 다리 각각이 담당하는 일을 한마디로 정리하면 이렇다. 청구는 시스템이 돈의 방향을 정하는 자리다. 누구에게, 얼마를, 어떤 주기로 청구할지를 결정한다. 수금은 정한 금액이 실제로 옮겨지는 자리다. 환불은 그 이동을 되돌리는 자리다.

각 다리에 해당하는 실패를 하나씩 붙이면 이런 그림이 된다. 청구 단계에서 proration을 잘못 계산하면, 고객은 이번 달에 두 번 결제되는 경험을 하게 된다. 수금 단계에서 웹훅을 잃으면, 결제는 됐는데 내 서비스는 그 고객에게 여전히 비활성 화면을 보인다. 환불 단계에서 상태를 갱신하지 않으면, 환불은 끝났는데 고객은 여전히 서비스 접속이 되는, 또는 접속이 안 되는

모순이 생긴다. 실패의 모양은 달라도, 원천은 하나다. 돈이 움직인 사실과, 시스템이 알고 있는 사실 사이에 틈이 생긴 것이다.

각 다리의 실패는 다음 다리로 전파된다. 청구 단계의 proration(중간 정산) 실수는 수금 단계에서 중복 청구가 된다. 수금 실패를 못 잡아내면 서비스는 계속 돌아가고, 그 대가는 나중에 환불과 분쟁 청구로 돌아온다. 그래서 돈 시스템의 첫 설계 과제는 결제가 완벽해야 한다는 것이 아니다. 다리가 하나 부러졌을 때 알 수 있게 만드는 것이 핵심이다.

1.3 혼자면 더 위험한 이유

이 실패들은 팀이 크다고 사라지지 않는다. 오히려 1인 개발자에게는 더 치명적인 구조가 있다.

팀이 있으면 밤에 웹훅이 죽어도 온콜이 있고 대조 작업을 따로 돌릴 엔지니어가 있고 은행 입금과 매출 보고서를 맞보는 사람이 따로 있다. 혼자면 그 역할이 전부 나다. 새벽에 고객이 보낸 환불 요청 메일은 내 알람이고, 정산 불일치의 첫 탐지자는 내가 보는 은행앱이다. 돈 시스템의 감시 장치가 사람 수만큼 줄어드는 자리가 바로 여기다.

그래서 혼자서는 두 가지 규율이 더 중요해진다. 첫째, 자동화되지 않은 감시기는 없다. 내가 눈으로 볼 수 없는 주기로 돌아가는 대조 작업은 없는 것과 같다. 둘째, 로그를 읽을 수 있어야 한다. 장애가 났을 때 PSP 화면을 열어서 눈으로 대조하는 수준이 아니다. 내 이력에서 원인을 10분 안에 찾을 수 있어야 한다. 이 두 가지는 4장의 웹훅과 대조 설계로 이어진다.

1.4 이력을 먼저 쓰라

첫날부터 몸에 익혀야 하는 규율이 하나 있다. 이벤트를 먼저 적고 상태를 나중에 바꾼다는 것이다.

돈과 관련된 모든 일, 청구, 환불, 상태 전이는 모두 상태를 적용하기 전에 로그 테이블에 한 줄을 쓴다. 이 로그를 돈 이력(ledger)이라고 부르자. 최소 필드는 이렇다.

- 이벤트 식별자, 고유
- 타입, 예를 들어 invoice.paid, refund.succeeded
- 시각
- 금액과 통화, 항상 정수, 통화의 최단위 단위
- 전 상태와 후 상태
- 외부 참조, PSP의 식별자
- 출처, 웹훅, 대조, 수동

한 줄이 실제로 생긴다.

```
id: evt_00217
type: invoice.paid
ts: 2026-07-08T09:14:03Z
subscription: sub_42
amount: 33000 KRW
prev_state: open
new_state: paid
source: webhook
psp_ref: in_9f2c
```

왜 이렇게까지 하느냐. 세 가지 이유다.

첫째, 웹훅은 순서를 지키지 않고 중복으로 도착한다. 이벤트를 도착 순서대로 처리하면 상태가 꼬인다. 이력을 먼저 적으면 꼬인 상태를 로그에서 다시 유도할 수 있다. 이력은 재현 가능한 유일한 진실원이 된다.

둘째, 사람이 있기 때문이다. 고객이 내가 돈을 뱉걕는데 서비스는 안 켜진다고 할 때, 그날의 일은 기억나지 않는다. 이력은 기억해준다. sub_42의 이벤트를 시간순으로 읽으면 대부분의 고객 분쟁은 로그 읽기

단계에서 끝난다.

셋째, 대조 때문이다. 4장에서 다루는, 내 데이터베이스와 PSP의 데이터를 비교하는 밤 작업에는 내가 처리한 이벤트 목록이 필요하다. 그 목록이 이력이다.

실무적 경고가 하나 있다. 이벤트 기록과 상태 변경은 같은 트랜잭션에 넣을 것. 상태를 먼저 바꾸고 로그를 나중에 쓰면 로그 없는 상태가 생긴다. 로그를 먼저 쓰고 상태 변경 도중 프로세스가 죽으면 로그만 있고 상태는 제자리다. 후자는 로그를 다시 재생하면 되지만 전자는 상태가 어떻게 바뀌었는지를 알 길이 없다. 돈 시스템에서는 항상 다시 쓸 수 있는 쪽을 고른다.

1.5 상태는 사실이다

구독 상태 머신의 최소 구성은 네 개다. active, past_due, canceled, unpaid.

상태	의미	서비스 효과
active	정상 청구 진행 중	이용 가능
past_due	청구 실패, 재시도 중	유예 정책에 따름
canceled	갱신 중단	유료 기간까지 이용
unpaid	재시도 소진	이용 중단

이 상태들에 대한 규칙은 세 가지다.

첫째, 상태를 움직이는 것은 이벤트뿐이다. 사용자가 버튼을 누르고 API를 부르는 것은 상태 전이 요청일 뿐이다. 실제 전이는 청구 이벤트(웹훅)나 대조 작업이 수행한다. API의 일은 받은 것을 답하는 것.

둘째, past_due에서는 반드시 나가야 할 길이 있어야 한다. 고객이 카드를

갱신하고 재시도가 성공하면 active로 돌아온다. 이런 경로가 없으면 past_due는 unpaid 앞의 대기실이 된다. 카드가 만료됐을 뿐, 해지할 의도는 없던 고객을 여기서 잃는 것이다.

셋째, 설명할 수 없는 상태를 만들지 마라. 상태가 하나 늘면 전이 조합이 늘어나고 각 조합에 돈은 어떻게 되느냐는 답이 필요하다. trial, paused 같은 상태를 많은 시스템이 추가하고 줄인다. 시작은 최소 네 개에서 하고 전이의 돈 효과가 분명히 적을 수 있을 때만 상태를 늘리는 편이 좋다.

가장 자주 빠지는 상태가 past_due다. active와 canceled만으로 만드는 두 상태 세계에서 청구가 실패하면, 즉시 canceled로 갈 것인지(active 상태로 영원히 남을 것인지)를 고르게 된다. 즉시면 고객은 카드 갱신 기회조차 없이 서비스를 잃고, 지원 티켓과 이탈이 생긴다. 영원히면 이 장의 첫 장면에서 본 수익 누수가 된다. 세 번째 상태가 없는 자리에서 돈이 조용히 새어나가는 것이다.

1.6 무언가 부러졌을 때 보는 순서

돈 관련 불평이 들어오면, Support 티켓이든 메일이든, 보는 순서가 있다. 이 순서를 지키면 대부분의 사고는 30분 안에 원인을 찾는다.

1. 해당 구독의 이력을 시간순으로 읽는다. 어떤 이벤트가 언제 왔고 무엇이 빠졌는지 보인다.
2. 빠진 이벤트가 있으면 PSP 쪽에 그 이벤트가 있는지 확인한다. PSP에 있고 내 이력에는 없으면, 웹훅이 죽은 구간이다.
3. PSP에도 없으면, 청구 자체가 안 된 것이다. 청구 일지와 dunning 상태를 본다.
4. 돈을 움직인 결과가 다르면, 환불과 분쟁 이벤트를 이력에서 찾는다.

이 순서가 통하는 전제는, 이력이 정직하게 쌓여 있다는 것이다. 1단계에서 이력 읽기가 불가능한 시스템은, 2단계부터의 모든 단계가 눈대짐이 된다.

돈 시스템의 운영 능력이 이력 하나의 품질에 걸려 있다는 것이, 이 장이 말하고자 한 바의 실용적 결론이다.

1.7 실제로 부러지는 자리

돈 시스템이 실제로 무너지는 자리를 모으면 목록은 길지 않다.

- 플랜 변경 시 proration 오류. 매번 몇 천 원의 오차가 쌓여 환불과 분쟁 단계에서 불일치가 된다. 2장.
- 실패한 청구에 재시도 스케줄이 없다. 언제 포기할지 모르면 영원히 재시도하거나, 너무 일찍 포기한다. 2장.
- 결제 요청의 중복 청구. 멱등 키 부재가 원인이다. 3장.
- 돈은 움직였고 상태는 그대로. 환불은 됐는데 구독이 활성이거나, 그 반대. 3장.
- 웹훅 유실. 이벤트가 사라진 것을 잡는 작업이 없다. 4장.
- 징수하지 않은 세금. 가격은 정해졌고 관할은 생각하지 못했다. 4장.

여섯 가지 모두 나중에 한꺼번에 터지지 않는다. 청구 오류는 다음 청구 주기에, 환불 누락은 다음 정산일에, 웹훅 유실은 다음 대조 시점에 조용히 쌓인다. 터지는 순간이 아니라 쌓이는 기간이 문제다. 매출이 조금씩 줄어드는 것이 눈에 띄지 않는 것과 마찬가지다.

이 실패들의 공통 형태를 주목할 것. 시스템이 계속 잘 돌아가는 동안 일어난다. 예외도, 에러 메시지도 없다. 돈 시스템 실패 공간의 특징이 이렇다. 큰 소리로 부러지지 않고 조용히 부러진다. 그래서 돈이 지나가는 자리의 개발자 일은 모든 에러를 막는 것이 아니다. 모든 에러가 하루 안에 보이게 만드는 것. 이력, 상태 머신, 대조 작업. 이 책의 모든 도구는 그 한 마디를 위해 있다.

2 구독 청구 설계: 날짜, 상태, 실패한 카드

청구 날짜는 단순해 보인다. 매달 1일에 3만 원을 청구하면 된다. 그런데 고객이 달 도중에 플랜을 바꾸고 카드가 실패하고 고객이 중도 해지하면, 날짜는 단순하지 않다. 이 장은 세 가지를 다룬다. proration을 어떻게 계산하는지, 실패한 청구의 재시도 스케줄을 어떻게 짜는지, 실패한 카드를 어떻게 회복하는지.

2.1 청구 날짜: 두 개의 앵커

청구 날짜를 정하는 방식은 두 가지.

첫째는 시작일에 앵커를 박는 방식. 고객이 17일에 가입했다면 매달 17일에 청구한다. PSP가 이 날짜를 anchor date라고 부른다. 특징은 모든 고객의 주기가 다르다는 점. 어떤 날을 잡으면, 어떤 고객의 주기는 2일밖에 안 남았고 다른 고객의 주기는 28일 남는다.

둘째는 달력에 고정하는 방식. 모두 매달 1일에 청구한다. 첫 달의 주기는 짧다. 17일에 가입한 고객의 첫 청구는 다음 달 1일에, 15일만 지나면 온다. 특징은 모든 고객이 동기화되어 지원 업무와 매출 예측이 쉬워진다는 점. 대신 첫 주기가 정규 주기와 길이가 달라, proration 로직이 첫 주기를 따로 다뤄야 한다.

1인 개발자에게 기본값은 앵커 방식. PSP의 기본 동작이기도 하고 proration 로직이 균일하다. 언제나 현재 주기의 남은 일수로 계산하면 되니까. 고정 방식은 B2B 계약에서 월 단위 청구서를 요구받을 때 고려할 가치가 있다. 그 전에는 첫 달을 위한 별도 분기를 만드는 데 드는 버그 수만큼 손해다.

2.2 proration: 단순해 보이는 계산

상황부터 잡자. 3만 원 월정액 플랜의 고객이 해당 월 10일째 되는 날에 5만 원 플랜으로 업그레이드한다. 청구 주기는 30일.

proration의 원리는 이것. 옛 플랜의 남은 기간을 정산하고 새 플랜의 남은 기간을 청구한다. 이미 낸 돈은 신용(credit)으로 돌려준다.

- 옛 플랜 21일 잔여: 30000 x 21/30 = 21000원을 credit
- 새 플랜 21일분: 50000 x 21/30 = 35000원을 청구
- 10일 어를 순청구액: 35000 - 21000 = 14000원
- 다음 청구일(다음 달 10일): 50000원

다운그레이드는 거꾸다. 옛 플랜 credit이 새 플랜 비용보다 크면, 그 차이는 차기 청구에서 뺄 credit 잔액이 된다. 여기서 정책 선택이 하나 있다. credit을 이월한다(PSP 기본), 환불한다(드물고 수수료는 못 온다), 버린다(고객이 항의한다). 1인 개발자에게는 이월을 권한다. 단순하고 돈이 움직이지 않아서 이력이 깨끗하다.

일수 계산 관례는 처음에 한 번 정하고 바꾸지 않는 것. 잔여 일수를 달력 일수로 셀 것인지, PSP 내부 관례를 따를 것인지(PSP마다 달의 일수로 계산하는 곳도 있다). 실무 규칙은 이렇다. proration 결과 금액은 PSP가 계산한 값을 쓰고, 내 계산과 대조하는 일은 대조 작업(4장)에게 맡긴다. 불일치가 시스템적으로 나타난다면, 관례를 맞춰야 할 쪽이 어디인지 찾으면 된다.

테스트해야 하는 엣지 케이스는 네 개.

- 주기의 마지막 날에 플랜 변경. 잔여 일수가 0이거나 1이고 순청구액이 0원이거나 음수일 수 있다.
- past_due 상태에서 변경. 실패한 청구 위에 proration을 얹는 일이다. 실패한 청구서에 공제할지, 새 청구서를 만들지 정해야 한다.

- credit 잔액이 다음 청구액을 넘는 다운그레이드. credit이 그다음 주기까지 이어간다.
- 주기 중 해지. 남은 일수 환불 여부는 환불 정책의 문제(3장)다. proration 계산과 섞지 않는 것. 둘을 한 함수에 넣으면 둘 다 틀어진다.

2.3 실패 청구 파이프라인: 최종 절단이 있는 스케줄

청구가 실패한 순간, 구독은 past_due로 들어간다. 여기서부터 시스템이 재시도 스케줄을 돈다. 스케줄 자체는 자유다. 다만 형태가 중요하다. 권하는 기본값은 이렇다.

시점	액션	상태
청구일	1차 청구	실패 시 past_due
+3일	2차 청구, 안내 메일	past_due 유지
+7일	3차 청구	past_due 유지
+14일	최종 청구	실패 시 unpaid

이 형태 뒤의 논리를 적어두자. 초기 재시도는 간격이 짧다. 대부분의 실패는 일시적이다. 잔액 부족, 카드 만료, 은행쪽 오류. 고객은 며칠 안에 고치는 경우가 많다. 후기 재시도는 간격이 길다. 거기까지 남은 고객은 카드가 정말 죽었거나, 내 의지가 없는 쪽이다. 짧은 간격의 재시도는 거기서 노이즈만 추가한다. 그리고 최종 절단이 반드시 있어야 한다. 절단이 없으면 past_due가 상시 상태가 되고 이 고객이 돈을 내고 있느냐는 질문에 답할 수 없게 된다.

재시도는 같은 청구서에 대한 새 시도다. 이력은 시도들을 기록한다.

```
invoice: inv_1024, 50000 KRW
attempt 1: 2026-07-10 card_declined
attempt 2: 2026-07-13 card_declined
```

```
attempt 3: 2026-07-17 paid
```

돈 효과는 하나다. inv_1024가 지급 완료된 것. 시도는 세 번. 이 구분이 환불 때 중요해진다. 환불은 청구서에 한다. 시도에 하지 않는다.

이 기간 동안 서비스는 유지할지 중단할지가 유예 기간(grace period) 정책이다. 이건 비즈니스 결정이지 엔지니어링 결정이 아니다. 다만 문서로 적어 두고 코드에 반영해야 한다. 흔한 기본값: 유료 기간이 끝날 때까지는 서비스를 유지하고, unpaid가 되는 순간 중단한다. 첫 실패에 즉시 끊으면 고객은 카드 갱신 기회도 없이 서비스를 잃는다. 영원히 유지하면 1장 도입의 수익 누수가 된다. 중단 시점은 상태에 매긴다. 시간에 değil.

메일은 두 통이다. 첫 실패 때 청구 실패, 카드 갱신 필요라는 메일에 PSP의 호스티드 카드 갱신 링크를 넣는다. 이 링크가 핵심이다. 고객이 PSP 화면에서 카드를 바꾸고 내 데이터베이스에는 카드 번호가 닿지 않으며 갱신 성공에 웹훅이 온다. 최종 절단 직전의 재시도 전에 마지막 기회를 알리는 메일을 한 통 더 보낸다. 그 이상은 보내지 않는 것. 시스템이 갱신해 주세요라는 메일을 너무 많이 보내면, 고객은 열지 않게 되고 마지막 한 통이 힘을 잃는다.

2.4 카드 회복: 죽음이 아낭라아 이벤트

가장 흔한 실패는 카드 만료다. 흐름은 이렇다. 고객이 PSP 포설에서 카드를 갱신하고 웹훅이 온다(카드 갱신 이벤트, 또는 새 청구 시도 이벤트). 재시도 청구가 성공하고 상태가 past_due에서 active로 돌아온다.

중요한 설계 포인트는 이 흐름이 고객이 요청하지 않아도 돌아가야 한다는 것이다. 여기서 빠지기 쉬운 함정이 하나 있다. PSP에도 실패 청구 재시도(dunning) 기능이 있고, 나도 위에서처럼 내 스케줄을 만들면, 두 재시도 엔진이 동시에 돈다. 한 번의 실패에 PSP가 재시도하고 내가 재시도하면, 고객은 같은 주기에 두 번 청구를 받을 수 있다. 중복 청구의

교과서적 원인이다. 고르는 방식은 하나다. PSP 자동 재시도에 맡기면 내 스케줄은 PSP가 소진된 뒤의 폴백 정도로만 두고, 내 스케줄을 만들면 PSP 자동 재시도를 끈다. 둘 다 켜 놓는 것은 우연이다.

데이터 보존 정책도 상태에 매긴다. 구독이 canceled나 unpaid가 되면, 데이터는 즉시 삭제하지 않는다. 정해진 기간을 보존하고 그 뒤에 아카이빙한다. 기간은 정책 선택이다. 30일, 90일, 법이 요구하는 기간. 이유가 두 가지다. 하나는 unpaid 고객이 새 카드로 돌아올 수 있고 데이터를 지워두면 재구독이 신규 가입이 된다. 다른 하나는 거래 기록의 보관 의무가 있다는 것. 언제 지울지는 상태 기반의 예약 작업이 답한다. 개발자가 일할 때 기분으로 답하지 않는다.

2.5 청구를 테스트하는 법

돈 시스템의 테스트는 실카드로 하지 않는다. PSP의 테스트 모드(개발 모드)는 가짜 청구만 만들고 실제 돈은 움직이지 않는다. 여기서 할 수 있는 일이 많다.

- 성공, card_declined, insufficient_funds, processing_error 등 실패 코드별 테스트 카드 번호를 PSP가 제공한다. 각 코드별로 내 시스템이 어떻게 반응하는지 확인한다.
- 실패 청구가 발생하면 dunning 스케줄이 실제로 도는지, 상태가 past_due로 옮겨가는지, 메일이 나는지 본다.
- 카드 갱신 웹훅을 테스트 포설에서 만들어 보면, past_due에서 active로 돌아오는 경로가 검증된다.
- 웹훅 엔드포인트가 배포 환경에서 서명 검증을 통과하는지, 멱등 처리가 중복 이벤트를 버리는지 확인한다.

테스트에서 특히 값진 순간은 실패를 유도한 다음, 그 실패가 이력, 상태, 메일 세 곳에 일관되게 나타나는 것을 보는 것이다. 한 곳만 맞으면 나머지도 어긋나 있다는 신호다. 출시 전에 이 관측을 한 번이라도 해 두면, 실제

장애가 왔을 때 어디를 볼지 안다.

2.6 청구서는 구독이 아니다

두 개를 헷갈리는 엔티티가 있다. 구독과 청구서다. 구독은 긴 관계다. 상태, 플랜, 청구 날짜를 가진다. 청구서는 그 관계 안에서 일어나는 한 번의 돈 이동이다. 금액, 상태, 시도를 가진다.

구독에 '이번에 낸 금액'을 직접 붙여 두고 청구서 엔티티를 생략하면, proration에서 막힌다. 이번 주기는 얼마였고 지난 주기는 얼마였는지를 답할 곳이 없기 때문이다. 환불도, 분쟁도 청구서에 붙는 것이지 구독에 붙는 것이 아니다.

실무적 최소 구성은 이렇다. 구독 테이블은 상태와 플랜을 갖고 청구서 테이블은 금액과 상태와 속한 주기를 가진다. 청구서의 상태는 open, paid, void, uncollectible 정도면 충분하다. 이력은 청구서 상태 전이를 기록한다. 구독 상태 전이는 아니다. 두 엔티티를 섞으면, 이번 달에 실제로 얼마가 청구됐느냐는 질문에 답할 쿼리가 사라진다.

2.7 멱등 재시도: 돈 효과는 한 번

수동 재시도 경로가 마지막이다. 관리자 화면의 이 청구 다시 시도 버튼, 예약 작업, 지원 담당자가 트리거하는 재청구. 각각은 결제 요청이고 결제 요청에는 중복의 세 경로가 있다. 내 재시도, 게이트웨이 재시도, 이중 클릭.

규율은 이것이다. 결제 의도 하나에는 키 하나. 같은 청구서를 수동으로 다시 시도하면, 원래 의도의 키를 쓴다(또는 청구서 식별자에서 파생한다). 새로운 청구라면 새로운 키를 낸다. PSP는 같은 키에 같은 결과를 돌려주고 두 번 청구하지 않는다. 3장에서 메커니즘을 다룬다. 청구 쪽의 원칙은 이 한 줄이다. 돈 효과는 청구서에 속하고 시도는 로그에 속한다. 재시도가 두 번째 돈 효과를 만들게 두지 않는 것.

이 장이 끝나면, 구독에는 이런 것들이 있다. 청구 날짜, 엣지 케이스를 다룬 proration 규칙, 최종 절단이 있는 재시도 스케줄, 카드 회복 경로, 시도와 효과를 구분하는 이력. 다음 장은 돈이 실제로 오가는 레일로 나간다. 어떤 레일 위에 돈이 타는지, 어떻게 되돌려 보내는지.

3 결제 레일과 환불: 돈이 오가는 길

환불 요청은 돈 시스템이 진짜로 시험받는 순간이다. 청구는 결제 서비스 제공사(이하 PSP) 뒤로 숨길 수 있다. 환불은 내 코드, PSP, 고객의 카드라는 세 주첫가 같은 숫자에 동의해야 하는 자리다. 이 장은 네 가지를 다룬다. 레일 고르는 법, 중복 청구를 막는 멱등 키, 실제로 내 계좌에 들어오는 돈의 구조, 환불을 트랜잭션으로 다루는 법.

3.1 레일 세 가지

돈을 받는 방식은 세 가지다. 결제 서비스 제공사(PSP, Stripe 같은 글로벌 회사), 결제 대행사(PG, 토스페이먼츠나 포틀원 같은 국내 회사), 카드사나 은행과의 직결.

레일	강한 점	약한 점
PSP	해외 카드, 기능 전체	국내 결제 수단 제한
PG	국내 카드, 선불충전 수단	해외 카드, 기능 다양성
직결	완전한 제어	1인에게 비현실적

직결은 카드사나 은행과의 계약, 정산 시스템, PCI 컴플라이언스 부담이 필요하다. 1인 개발자가 짊어질 수 없는 수준이다. 그래서 실질적 선택은 앞에 두 가지, 또는 그 조합.

레일에서 사는 것은 결제 능력 하나가 아니다. 일괄 패키지다. 패키지의 구성 요소:

- 토큰화. 카드 번호는 PSP의 저장고에 있고 내 데이터베이스에는 토큰만 있다. PCI 자가진단 범위가 최소 수준(SAQ A)까지 줄어든다. 이 항목 하나만으로 수수료가 정당하다.

- 호스티드 체크아웃 또는 결제 요소. 결제 화면이 PSP의 도메인이나 iframe 안에 있다. 고객이 카드 번호를 입력하는 곳은 내 서버가 아니다.
- 실패 청구 관리. 카드 갱신 포설, 재시도, dunning 메일. 2장의 스케줄을 여기가 대신해 준다.
- 정산과 보고. 거둔 돈을 사이클대로 입금하고 대조에 쓸 데이터를 API와 화면으로 제공한다.
- 세금 계산(선택). 고객의 위치에 따라 세율을 계산해 준다. 4장의 세금 경계가 여기서 시작된다.
- 환불과 분쟁 API. 환불과 chargeback(분쟁 청구)을 이벤트로 처리해 준다.

선택 기준은 단순하다. 고객이 어디에 있느냐. 한국에서만 파는 거라면 국내 PG. 국내 카드와 선불충전 수단을 다루고 원화로 정산하고 지원이 한국어다. 첫날부터 해외 고객이 있다면 PSP. 둘 다라면 PSP와 PG를 같이 쓴다(또는 국내 PG 연동이 있는 PSP). 레일에 대한 생각은 여기서 끝내는 편이 좋다. 중요한 것은 레일 위의 이벤트와 상태 설계이며, 그건 어느 레일에서나 같다.

3.2 결제 요청의 멱등 키

결제 요청은 중복의 세 경로를 가진다. 내 재시도(타임아웃이라 다시 보냄), 게이트웨이나 프록시의 재시도(모르고 켜둔 설정), 사용자의 이중 클릭(화면이 멈춘 듯하면 사람은 반드시 다시 누름).

규율은 이것이다. 결제 의도 하나에는 키 하나. 클라이언트가 UUID를 만들어 요청에 idempotency-key로 보낸다. PSP는 같은 키에 같은 결과를 돌려주고 두 번째 청구를 하지 않는다.

```
POST /v1/payments
idempotency-key: 7c1e9a2b-...
```

```
(1차 요청, 타임아웃, 응답 없음)

POST /v1/payments
idempotency-key: 7c1e9a2b-...

(2차 요청, 200 OK, 같은 결제 식별자 반환, 새 청구 없음)
```

키가 없으면 타임아웃은 미스터리다. 첫 요청이 청구됐는지 알 수 없다. 키가 있으면 타임아웃은 안전한 이벤트가 된다. 같은 키로 다시 보내면 된다. 1장에서 타임아웃은 실패가 아니라 정보 부재라고 했다. 멱등 키는 그 정보 부재 아래에서도 다시 보내게 해 주는 도구.

주의는 두 가지다. 키는 같은 PSP, 같은 엔드포인트 안에서만 유효하고 유효 기간이 있다(PSP 문서에 따라 보통 하루 전후 [추정]). 다른 결제 의도에 키를 재사용하면 안 된다. A 청구서의 키로 B 청구서를 만들면, PSP는 A의 결과를 돌려준다. 키는 의도의 신원이지, 절약을 위한 재사용 토큰이 아니다.

3.3 정산: 실제로 들어오는 돈

청구한 돈은 아직 내 것이 아니다. PSP가 거두고 보유하고 정산 사이클에 맞춰 입금한다. 사이클은 PSP와 계좌에 따라 다르다(일일, 며칠 단위, 주 단위 [추정]). 그래서 고객이 결제한 순간과 내 은행 계좌에 돈이 들어온 순간 사이에는 며칠의 간극이 있고, 그 간극에서 숫자는 맞지 않는다.

PSP는 거둬 든 돈에서 수수료, 환불, 분쟁 손실, 경우에 따라 보유금을 뺀다. 수수료 구조는 대개 일정 비율에 건당 고정 금액을 더한 형태다. 정확한 수치는 PSP와 계약에 따라 다르니 문서를 확인한다. 수준은 몇 퍼센트 안팎 [추정].

한 청구서의 여정을 숫자로 보자.

- 청구액: 33000원(30000원 + 부가가치세 10%)
- 수수료 약 4% [추정]: 약 1320원
- 입금액: 약 31680원
- 나중에 30000원 전액 환불 발생: 다음 사이클 입금에서 30000원 차감
- 해당 건의 수수료는 돌아오지 않는다 [추정]: 순수 손실 약 1320원

마지막 줄이 1인 개발자를 놀라게 하는 부분이다. 환불은 돈을 돌려주는 것이 아니라, 돈을 돌려주고 수수료를 잃는 것이다. 분쟁이 된 청구라면 더 나간다. 청구액에 분쟁 수수료 [추정]가 더해지고 진다면 금액은 회복되지 않는다. 그래서 환불 정책은 PSP에 붙는 수수료를 비용으로 계산해야 하고, 가격은 수수료율을 알아 두고 정해야 한다.

실무 습관 하나. 매달 장부 매출(내 데이터베이스의 paid 청구서 합계)과 실제 입금(PSP의 정산 리포트)을 맞본다. 차이는 수수료 더하기 환불 더하기 분쟁 더하기 정산 타이밍의 합이다. 차이가 설명이 된다면 시스템은 건강하다. 설명이 안 된다면 이력에서 빠진 기록이 있고 그것은 버그다. 우연이 아니다.

3.4 환불은 트랜잭션이다

환불은 상태가 있다. pending, succeeded, failed. 가장 흔한 설계 실수가 환불을 버튼으로 만드는 것이다.

환불 상태	의미	다음 단계
pending	제출 완료, 확인 대기	웹훅 대기
succeeded	되돌리기 완료	구독 상태 갱신
failed	되돌리기 실패	수동 처리

failed 환불은 실제로 일어난다. 카드가 만료됐고 계좌가 닫혔고 은행이

거절한 경우다. succeeded 환불도 즉시 끝나지 않는다. 카드에 며칠 만에 도착한다 [추정]. 그래서 규칙은 이렇다. 환불 요청은 이벤트이고 환불 결과도 이벤트이며 상태는 결과 이벤트로만 움직인다. 요청 순간에 상태를 refunded로 만들면, 일어날지도 모르는 사실을 기록한 것이 된다.

환불이 구독에 주는 효과는 미리 정한다. 전액 환불은 보통 현재 유료 기간을 끝낸다(또는 그 청구서를 무효화한다). 부분 환불은 다르다. 구독을 끝내지 않을 수 있다. 선택지는 두 개. 차기 청구에서 공제할 credit으로 돌리거나, 사용하지 않은 기간의 proration 금액을 돌려준다. 정책 선택은 비즈니스 판단이지만 코드에서는 별개의 경로여야 한다. 환불이라는 말 하나에 void와 credit 두 개가 숨어 있다.

분쟁(chargeback)은 다른 길이다. 고객이 내 시스템에 요청하지 않고 은행에 직접 이 청구를 인정하지 않았다고 주장한다. 나는 웹훅으로 알리고 응답 기간(며칠에서 몇 주간, PSP에 따라 [추정]) 안에 증빙을 낸다. 서비스 제공 기록, 약관 동의 로그, 연락 시도 내역. 진다면 금액에 분쟁 수수료 [추정]가 더해져 나간다. 여기서 중요한 숫자 하나. 제때 환불하는 비용이 분쟁의 기대 비용보다 낮은 경우가 대부분이다. 분쟁은 고객이 나에게 요청하지 않은 때의 절차다. 고객이 어렵다는 이유의 절차는 아니다.

3.5 환불 판단: 세 질문

환불 요청이 오면 순서대로 세 가지를 묻는다.

첫째, 법적 또는 계약적 의무가 있는가. 한국의 원결 계약에는 소정의 기한 내 철회권이 있고, 디지털 재화는 다운로드 또는 서비스 제공 개시점에서 철회가 제한되는 별도 규칙이 있다. 세부 규정은 서비스 형태에 따라 다르다. 의문이면 법률 상담이 답이다. 의무가 있다면 판단은 거기서 끝이다. 남는 질문은 처리 방법뿐이다.

둘째, 거절의 비용은 얼마인가. 분쟁(청구액 더하기 수수료 더하기 응답

업무), 부정 리뷰, 몇 주가는 지원 대화. 1인 개발자의 시간은 이 사업에서 가장 비싼 비용 항목이다. 3만 원 환불어로 몇 시간을 살 수 휟다면, 대부분의 경우 좋은 거래다.

셋째, 환불의 비용은 얼마인가. 이미 소비된 사용량(사용량 과금 성분이 있다면), PSP에 붙는 수수료, 그리고 선례. 누구나 요청하면 환불해주면, 환불 정책은 작동하지 않게 된다.

판단 프레임은 이렇다. 기한 내 첫 요청은 기본 환불. 이후는 세 질문으로 건별로. 판단 결과와 이유를 항상 이력에 기록한다. 다음에 같은 요청이 오면 일관성으로 답한다. 그리고 분쟁이 오면, 그 판단 이유가 곧 증빙이 된다.

구체적 예시 하나. 3만 원 월정액의 고객이 10일째 되는 날, 맞지 않는다며 환불을 요청했다. 선택지는 세 개. 전액 환불(제품 초기 단계라면 이탈 고객 하나에게서 배우는 가치가 3만 원보다 크다), proration 환불(30000 x 20/30 = 20000원, 남은 기간분), credit(차기 월 3만 원 공제). 제품 불확실성이 높은 초기에는 전액 환불이 자주 정답이다.

3.6 취소와 환불은 다르다

끝으로, 늘 뒤섞이는 두 개념을 갈라놓자.

취소는 향후 청구를 멈춘다. 고객은 유료 기간까지 서비스를 계속 쓰고 돈은 움직이지 않는다. 환불은 돈을 되돌린다. 보통 유료 기간을 끝내며 돈이 움직인다.

환불이라는 제목의 티켓의 상당수는 더 이상 청구되지 않게 해 주십시오라는 요청이다. 해지라는 요청 안에는 낸 돈을 돌려받고 싶다는 기대가 숨어 있는 경우도 많다. 그래서 UI에는 버튼이 두 개, 코드에는 경로가 두 개, 지원 답변의 첫 문장은 두 가지 중 무엇을 원하시녜는칠문이다. 이 구분이 흐려지는 순간, 환불 건수와 해지 건수가

섞이고 매출 예측은껴지고 정산 명세서에서 얼마가 돌아왔는지를 답할 능력을 잃는다.

3장이 끝나면 돈이 오가는 길은 정해졌다. 레일을 고르고 키로 중복을 막고 입금을 수수료로 설명하고 환불에 상태를 붙였다. 마지막 장은 상태를 움직이는 자리로 간다. 웹훅, 그리고 웹훅이 죽었을 때 나를 지켜 줄 대조 작업, 그리고 세금의 경계.

4 웹훅, 대조, 그리고 세금 경계

구독이 PSP에서 해지됐다. 고객이 PSP 화면에서 해지했고 PSP의 상태는 canceled가 됐고, 웹훅이 보내졌다. 그런데 그날 밤 내 서버는 배포 중이었다. 웹훅은 배포 도중에 도착해 500을 받았고, PSP는 재시도했다. 며칠을 재시도했다. 그리고는 포기했다.

결과는 이렇다. PSP는 청구하기를 멈췄다. 내 데이터베이스는 여전히 active다. 서비스는 계속 켜져 있다. 청구는 안 들어오고 고객은 쓰고 있다. 또는 그 반대: 청구는 계속되고 고객은 이미 해지했다고 믿고 있다. 분쟁이 온다.

이 이야기는 과장이 아니다. 웹훅 유실의 가장 흔한 모양이다. 이 장은 세 가지를 다룬다. 웹훅을 잃지 않고 처리하는 법, 잃어버린 것을 찾아내는 대조 작업, 그리고 세금의 경계.

4.1 API 응답은 진실이 아니다

돈 시스템에는 정보가 두 종류 있다.

첫째는 API 응답이다. 내 요청을 접수했다는 뜻이다. 청구 요청에 200이 돌아온다는 것은 청구하겠습니다라는 것이지, 돈이 움직였다는 것이 아니다. 카드 승인꼬 정산은 별개의 단계고 PSP 쪽의 구독 상태는 아직 움직일 수 있다.

둘째는 웹훅 이벤트다. 무언가가 일어났다는 뜻이다. invoice.paid, charge.refunded, customer.subscription.updated. 이 이벤트들이 돈의 사실이고 비동기로, 최소 한 번씩, 순서를 지키지 않은 채로 온다.

그래서 설계 규칙은 하나다. 내 구독 상태 머신은 이벤트로만 움직인다.

사용자 API(해지 버튼, 플랜 변경)는 접수했다는 답을 주고 실제 상태 전이는 뒤에 오는 이벤트(또는 대조)가 수행한다. 요청 핸들러가 직접 구독 상태를 바꾸는 코드 경로가 있다면, 그것은 구멍이다. 그렇게 되면 상태의 진실원은 둘이 된다. 내 쓰기와 이벤트의 쓰기. 둘은 반드시 어긋난다.

출처	의미	신뢰 방식
API 응답	요청 수락	상태 증거로 쓰지 않음
웹훅	돈/상태 이벤트	1차 진실
대조	스냅샷 비교	안전망

4.2 웹훅 처리의 다섯 규칙

규칙 1, 서명을 검증한다. 모든 웹훅은 시크릿 키로 만든 서명(HMAC)을 가진다. 검증이 실패하면 거부한다. 이것은 내 엔드포인트를 노려 invoice.paid를 위조하려는 자를 막는 것이다. 돈 시스템에서 검증되지 않은 이벤트는 위협이다.

규칙 2, 빠르게 2xx를 돌려주되, 느리게 처리한다. PSP의 재시도 타이머는 빠른 확인 응답을 못 받는 순간부터 돈다. 핸들러가 데이터베이스 쓰기와 메일 발송을 하고 30초 만에 답하면, PSP는 실패로 간주하고 다시 보낸다. 핸들러의 일은 읽기, 검증, 큐에 넣기, 200을 돌려주는 것뿐이다. 실제 처리는 워커가 한다.

규칙 3, 멱등적으로 처리한다. 전달은 최소 한 번이다. 중복은 정상이다. 처리한 이벤트 식별자를 저장해 두고 이미 있으면 건너뛴다. 여기서 결정적 한 줄: 처리 기록과 상태 변경은 같은 트랜잭션이다. 기록을 먼저 하고 상태 변경이 실패하면 이벤트는 사라진다. 상태 변경을 먼저 하고 기록이 실패하면 중복이 다시 올 때 두 번 적용된다. 한 트랜잭션이 둘 다의 답이다.

규칙 4, 순서 역전을 처리한다. 오래된 이벤트가 새로운 이벤트보다 늦게 올 수 있다. canceled 이벤트가 updated 이벤트 뒤에 도착하는 식이다.

보호는 이것이다. 적용 전에 이벤트의 상태나 시각을 현재 상태와 비교하고, 이벤트가 현재보다 오래됐으면 로그를 남기고 건너뛴다. 1장에서 상태가 앞으로만지행 상태 머신은 이 문제에 자연히 강하다.

규칙 5, 재시도에만 기대지 않는다. PSP는 실패한 웹훅을 간격을 늘리며 일정 기간 재시도한다(대개 며칠 [추정]). 그런데 내 엔드포인트가 그 기간보다 오래 죽어 있으면, 이벤트는 영영 사라진다. 재시도는 짧은 장애를 덮고 그 이상의 유실은 대조 작업만이 찾아낸다. 다음즐이.

핸들러의 윤곽, 의사코드로:

```
POST /webhooks
 1. 서명 검증, 실패면 401
 2. enqueue(event_id, payload)
 3. 200 반환

worker
 1. event_id 이미 처리됨이면 skip
 2. 이벤트가 현재 상태보다 오래됐으면 로그, skip
 3. 상태 변경 적용 (이력 먼저)
 4. 처리됨 표시 (3과 같은 트랜잭션)
```

4.3 대조 작업: 잃어버린 것을 찾는다

대조 작업은 스케줄로 돌아가는 비교다. PSP가 믿는 상태와 내가 믿는 상태를.

실행은 매일 밤(또는 몇 시간마다)에 한다. 시스템이 작다고 못 할 것이 없다. 주 1회여도 된다. 대조 작업이 찾아내는 것은 조용한 누수다.

비교 항목:

- 구독: 모든 구독의 상태(active, past_due, canceled, unpaid)를

PSP 목록과 대조

- 청구서: 이번과 지난 청구 주기의 금액과 상태
- 환불: 모든 환불 요청과 결과
- 분쟁: 내가 응답하지 않은 분쟁이 없는지

차이	흔한 원인	처리
상태 불일치	웹훅 유실, 순서 역전	PSP 기준으로 갱신, 로그
돈만 있고 기록 없음	유실된 웹훅	PSP 이벤트 조회 후 재생
환불 미반영	환불 후 상태 미갱신	상태 갱신
분쟁 무응답	분쟁 웹훅 무시	수동 응답

차이를 처리할 때 원칙이 하나 있다. PSP를 믿는 것. PSP의 데이터가 돈의 상태이고 내 것은 사본이다. 사본이 틀린 것이다. 다만 사본을 고치는 행위도 이력에 이벤트로 기록한다(출처: reconciliation). 그 기록이 나중에 상태가 왜 바뀌었는지를 설명하는 증거다.

대조 작업에 두 번째 용도가 있다. 감사 이력이다. 고객이 나는 결제했는데 서비스는 안 켜진다, 또는 해지했는데 또 청구됐다 하면, 이력과 대조 차이로 답한다. 대조 작업이 없으면 PSP 화면을 열어서 눈으로 대조하는 수준이다. 그것도 안 하면, 은행앱의 입금 내역과 지원 티켓을 옆에 붙여 놓고 맞추는 것밖에 없다. 후자는 발굴이다.

4.4 세금 경계

세금은 1인 개발자가 가장 자주 내 문제가 아니라고 판단해서 틀리는 부분이다. 문제의 모양을 보면 이해가 된다. 세금은 세 가지가 동시에 정하는 값이다. 파는 사람의 위치, 사는 사람의 위치, 재화의 종류. 디지톨 서비스는 그 자첫로 별도 규칙이 있다. 이 셋이 어느 세율이 적용되는지를 결정한다.

한국 안의 경우는 상대적으로 단순하다. 부가가치세는 10%다. B2C(개인 고객)는 대개 세액이 포함된 가격, 즉 세금 포함 정가를 쓴다. B2B(사업자 고객)는 다르다. 사는 쪽이 사업자등록번호를 가지면, 보통 세금과 분리된 세금 별도 영수증을 요구한다. 그 영수증에는 세금 신고 의무가 따라온다.

해외 고객이 하나 생긴 순간, 복잡도가 점프한다. 미국에는 연방 세일스가 없다. 각 주, 경우에 따라 시가 자첫 세율을 정한다(주에 따라 수준이 다르고 높은 쪽은 합산해서 약 10% 전후 [추정]). 유럽은 VAT 척계이고 cross-border 규칙이 따로 있다. 세율은 고객의 주소에 따라 달라지고, 시간에 따라 바뀌는 조회값이다.

실용적 경계는 네 가지다.

첫째, PSP의 세금 계산 기능을 처음부터 켜 두는 것. 주요 PSP는 관할 규칙과 세율 데이터를 함께 제공한다. 내 코드에 세율을 쓰면 썩는다. 세율은 바뀌고 관할은 바뀌고 내 코드는 그날의 스냅샷이기 때문이다.

둘째, 세금 계산은 제품의 기능이 되지 않게 하는 것. 청구서에 적용된 세율은 이벤트와 함께 도착하는 사실이다. 내 일은 그것을 이력에 기록하는 것.

셋째, 세무사를 끌고 오는 시점을 정하는 것. 신호는 이렇다. 첫 해외 판매, 첫 B2B 영수증, 연간 매출 규모가 세무 전문가의 판단을 필요로 하게 되는 때(국가에 따라 다르고, 전문가 판단이다). 일찍 끌고 오는 것이 나중에 쌓인 것을 정리하는 것보다 싸다.

넷째, 원 데이터를 남기는 것. 누구(고객 식별자), 어디(청구 주소, 국가), 얼마(세금 전, 세금, 세금 후), 무엇(세금 종류, 세율, PSP 참조). 세금 조사는 이 데이터에 대한 쿼리다. 이력이 있으면 한 시간에 답한다. 메일 쓰레기통과 기억에 있으면 몇 달에 답한다.

상황	누가 계산	언제 대응
국내 B2C	PSP 세금 기능	서비스 출시 전
국내 B2B	PSP 영수증 + 세무사	첫 계약 때
해외 판매	PSP 세금 기능 + 세무사	첫 해외 거래 때
세금 조회	세무사	데이터는 항상 준비

솔직한 보충. 이 절은 세무 자문이 아니다. 세금 규칙은 바뀌고 의무의 수준은 사업자 형태에 따라 다르다. 이 장의 포인트는 경계의 위치를 아는 것뿐이다. 경계는 우리가 계산하는 것을 멈추고 넘기는 것을 시작하는 선이다.

4.5 마무리: 돈이 지나가는 자리의 규율

이 책이 쌓아 온 규율을 한 장에 모으면 이렇다.

- 이력 먼저. 상태뱅갱천에 이벤트를 적고 같은 트랜잭션에.
- 상태 네 개. active, past_due, canceled, unpaid. 이벤트만 움직인다.
- 최종 절단이 있는 스케줄. 실패 청구는 영원하지 않다. past_due에서 나갈 길이 있다.
- 엣지를 테스트한 proration. 주기 마지막 날의 플랜 변경에 답이 있다.
- 결제 의도별 멱등 키. 타임아웃은 실패가 아니라 안전한 재전송이다.
- 트랜잭션인 환불. pending, succeeded, failed. 판단 이유는 이력에.
- 서명 검증된 웹훅. 빠른 확인, 멱등 처리, 순서 역전 보호.
- 스케줄로 도는 대조 작업. 차이는 PSP를 믿고 고치는 행위는 기록한다.
- 레일에 맡긴 세금. 세율은 조회값이다. 원 데이터는 이력에.

돈 시스템의 목표는 에러가 없는 세계가 아니다. 에러가 하루 안에 보이는 세계다. 무언가 부러졌을 때, 이력은 정렬되어 있고 대조 차이는 화면에 있고 돈은 조용히 새어나가지 않는다. 돈이 지나가는 자리의 개발자 일은 거기서 끝난다.