M04-01 · GIBALJA VISUAL MANUAL

백엔드가 처리하는 일 구분하기

> **한 문장 목표:** 화면의 버튼 한 번이 서버 안에서 어떤 판정·상태 변경·후속 작업·응답으로 이어지는지 그리고, 성공·거절·실패·중복·지연마다 확인할 증거를 말할 수 있습니다.

난이도	개념	실습	셀프 테스트	최종 산출물
Level 1	60분	60분	15분	처리 지도, 책임표, 오류·비동기 계약

"주문 버튼을 누르면 DB에 저장합니다"는 중간 단계가 너무 많이 빠진 설명입니다. "서버가 요청 형식을 다시 검증하고, 로그인한 사용자가 이 주문을 만들 권한이 있는지 확인하고, 재고 규칙을 판정한 뒤 하나의 트랜잭션으로 주문을 저장합니다. 결제는 800ms 안에 결과가 없으면 rollback하며, 성공 시 영수증 작업을 queue에 넣고 주문 ID와 job ID를 201로 반환합니다"라고 말해야 구현과 검수가 같은 장면을 봅니다.

그림 1. 백엔드 처리의 여덟 문. 이 분류는 복잡한 화면 뒤 책임을 빠르게 찾기 위한 YEONCORE 학습 프레임이며 보편적인 코드 계층 표준은 아닙니다.

1

접수

method·path·body

2

구조 검증

형식·필수·범위

5

저장

transaction·commit

6

외부 연동

timeout·retry

화면 성공 = 서버의 모든 후속 작업 완료

3

신원·권한

누가 무엇을

4

업무 규칙

지금 허용되는가

7

비동기

queue·event·job

8

응답·관측

status·trace·log

가 아니다 · 완료 경계를 계약으로 정한다

1. 이 PDF를 공부하는 방법

1회차 · 여덟 문 익히기 · 15분

그림 1을 보며 `접수 → 구조 검증 → 신원·권한 → 업무 규칙 → 저장 → 외부 연동 → 비동기 → 응답·관측` 을 소리 내어 읽습니다. 각 문에서 "통과 증거"와 "멈출 때의 응답"을 한 가지씩 말합니다.

2회차 · 경계와 완료 뜻 구분하기 · 30분

클라이언트와 서버의 신뢰 경계를 확인합니다. `201 Created` , `202 Accepted` , `commit` , `job completed` 가 각각 무엇까지 완료했다는 뜻인지 구분합니다.

3회차 · 아홉 시나리오 실행하기 · 35분

백엔드 처리 추적 실습기에서 정상·검증 실패·미인증·권한 없음·업무 충돌·DB rollback·외부 timeout·비동기·중복 요청을 실행합니다.

4회차 · 실제 기능 한 개 설계하기 · 40분

백엔드 처리 지도 템플릿에 교육 신청·문의 접수·문서 승인 중 하나를 적습니다. 각 단계의 입력, 판정 주체, 상태 변경, 오류, 관측 ID를 채웁니다.

5회차 · 셀프 테스트 · 15분

정답을 가리고 12문제를 풉니다. 틀린 문제는 `신뢰 경계` , `판정 구분` , `완료 경계` , `실패 증거` 중 원인을 표시합니다.

> **학습 완료 기준**
> "서버 오류가 났습니다"를 "객체 권한과 재고 규칙은 통과했지만 주문 INSERT 뒤 DB 오류가 발생해 트랜잭션이 rollback되었습니다. 주문 수와 재고는 이전 값이고, 응답은 503 problem type `/problems/storage-unavailable` , 발생 건은 `prb-8f31` , 같은 요청 추적은 `trace_id` 로 찾습니다"처럼 설명할 수 있습니다.

2. 백엔드는 화면 뒤의 권위 있는 결정을 맡습니다

브라우저는 화면을 보여 주고 빠른 피드백을 줍니다. 백엔드는 믿을 수 없는 요청을 받아 누가 무엇을 할 수 있는지 판정하고, 여러 상태 변경을 일관되게 수행하고, 다른 시스템에 일을 맡기고, 결과와 증거를 남깁니다.

```
사용자 클릭
  → HTTP 요청
  → 서버의 판정과 상태 변경
  → HTTP 응답
  → 화면 상태 변화
  → 필요하면 비동기 후속 작업
```

2.1. 기능보다 책임을 묻습니다

약한 질문	책임을 드러내는 질문
이 API는 무엇을 하나요?	어떤 입력을 믿지 않고 누가 최종 판정하나요?
DB에 저장하나요?	어떤 변경이 함께 성공해야 하며 실패하면 무엇을 되돌리나요?
외부 API를 호출하나요?	언제까지 기다리고 어떤 실패만 몇 번 다시 시도하나요?
200을 주나요?	이 응답은 접수·저장·최종 완료 중 무엇을 보장하나요?
로그가 있나요?	이번 사용자 건을 어떤 ID로 trace·log·record에서 연결하나요?

2.2. 여덟 문은 진단 순서입니다

실제 코드가 반드시 여덟 폴더나 여덟 함수로 나뉘어야 한다는 뜻이 아닙니다. 프레임워크·서비스 규모·도메인에 따라 middleware, controller, application service, domain model, repository, worker가 다르게 배치됩니다. 이 교재의 여덟 문은 **빠진 책임을 찾는 검수 질문**입니다.

문	핵심 질문	통과 증거	대표 중단
접수	읽을 수 있는 요청인가	request·trace ID	400·413·415
검증	구조와 의미가 유효한가	검증 결과	400·422 등 계약값
신원·권한	누구이며 이 행동이 허용되는가	principal·policy decision	401·403

문	핵심 질문	통과 증거	대표 중단
업무 규칙	현재 상태에서 허용되는가	rule decision	409 등
저장	함께 성공할 변경인가	commit·record ID	rollback·5xx
외부 연동	의존 서비스 결과를 어떻게 다룰까	attempt·dependency result	timeout·fallback
비동기	나중에 할 일을 어떻게 추적할까	job·event ID	failed·dead letter
응답·관측	사용자와 운영자가 무엇을 알까	status·problem·trace	모호한 성공·추적 불가

30초 확인

“주문 생성 성공”이 무엇을 뜻하는지 말해 봅니다. 주문 record commit, 결제 승인, 영수증 발행, 이메일 도착은 서로 다른 완료 지점일 수 있습니다.

3. 클라이언트와 서버 사이에 신뢰 경계를 긋습니다

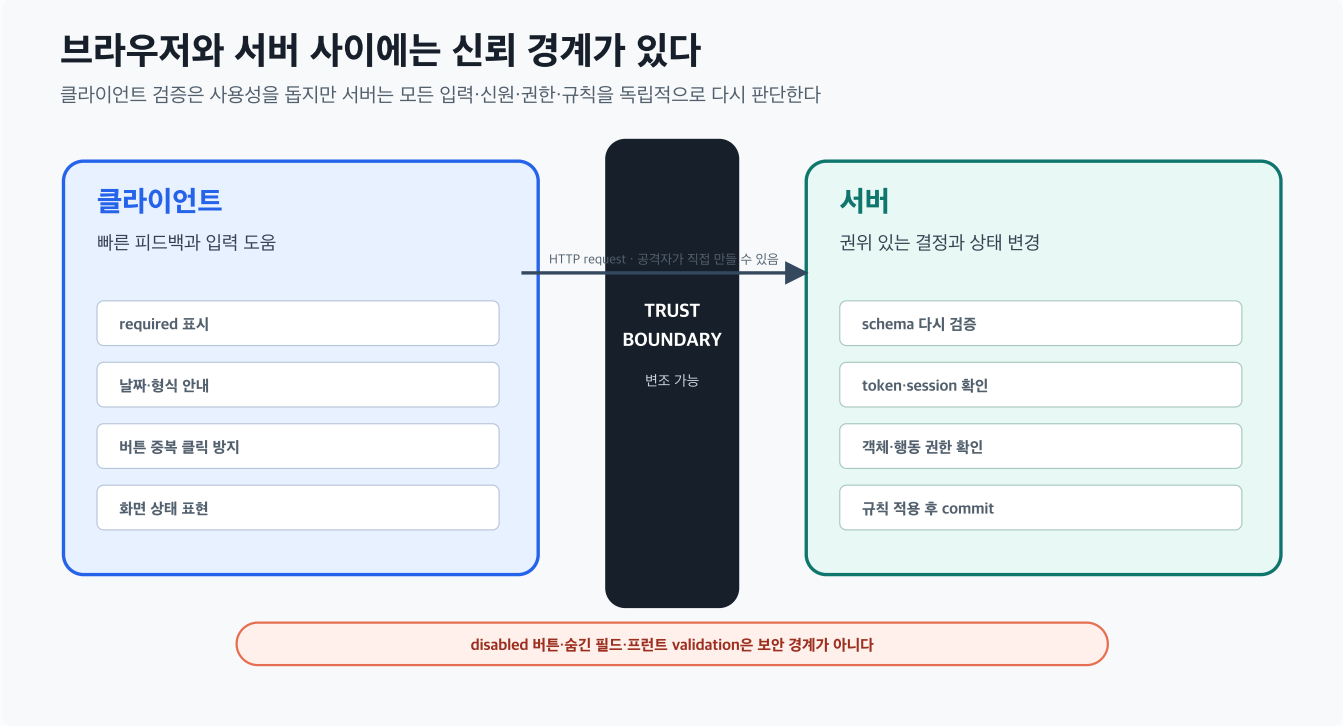


그림 2. 프런트 검증은 친절한 사용 경험이고 서버 검증은 권위 있는 판정입니다. 목적이 다르므로 둘 다 필요합니다.

클라이언트

빠른 피드백과 입력 도움

- required 표시
- 날짜·형식 안내
- 버튼 중복 클릭 방지
- 화면 상태 표현

HTTP request · 공격자

TRU
BOUN

변조

disabled 버튼·숨긴 필드·프런트

validation은 보안 경계가 아니다

브라우저의 disabled 버튼, 숨긴 input, JavaScript validation은 사용자를 돕지만 공격자는 브라우저 없이 HTTP 요청을 직접 만들 수 있습니다. OWASP Secure Coding Practices는 신뢰할 수 있는 시스템, 즉 서버 쪽에서 입력을 검증하고 클라이언트가 보낸 모든 데이터를 처리 전에 확인하도록 안내합니다.

3.1. 화면에서 막았다는 말은 서버 증거가 아닙니다

```
{
  "ownerId": "other-user",
  "quantity": -10,
  "role": "admin",
  "price": 1
}
```

화면에 `role`이나 `price` 입력란이 없어도 요청 본문에는 넣을 수 있습니다. 서버는 허용할 필드만 받아들이고, 가격·소유자·권한처럼 권위 있는 값은 서버 상태와 인증 문맥에서 결정해야 합니다.

3.2. 두 쪽의 검증 목적

항목	프런트	백엔드
필수 입력	즉시 표시·초점 이동	누락 요청 거절
형식	오타를 빠르게 알림	변조·직접 호출 포함 재검증
권한	불가능한 버튼을 숨김·비활성	모든 객체·행동에서 최종 판정
중복	버튼 잠금·진행 표시	같은 효과가 반복되지 않게 보호
오류	고칠 위치와 다음 행동 표시	안정적인 오류 종류와 수정 정보 제공

피해야 할 문장
“프런트에서 이미 검증했습니다.” 서버는 브라우저가 어떤 화면을 보여 줬는지, 버튼이 잠겼는지, 사용자가 정상 경로로 왔는지 믿을 수 없습니다.

4. 요청 접수에서 처리 문맥을 만듭니다

HTTP는 클라이언트가 request message를 보내고 서버가 status code를 포함한 response message를 보내는 요청·응답 프로토콜입니다. 백엔드는 먼저 method·target·headers·content를 읽고 이번 처리를 묶을 문맥을 만듭니다.

```
POST /orders HTTP/1.1
Content-Type: application/json
Authorization: Bearer [redacted]
Idempotency-Key: ord-260715-001
Traceparent: 00-4bf92f...-00f067...-01

{"productId":"COURSE-AI-01","quantity":1}
```

4.1. 접수 단계의 체크

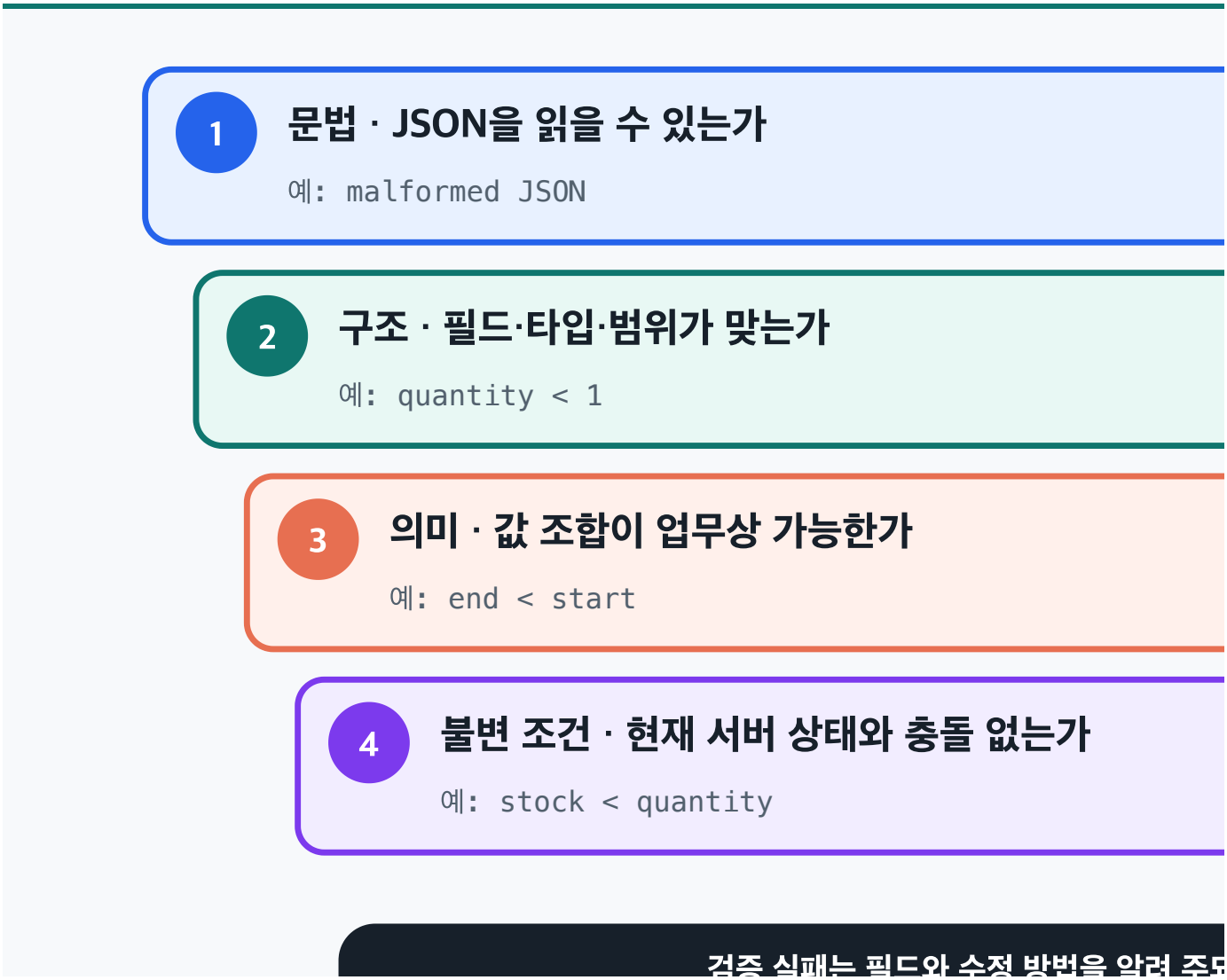
질문	예시 증거
지원하는 method·media type인가	`POST`, `application/json`
body를 읽을 수 있는가	parse 성공·400
크기·시간·자원 한도 안인가	content length·deadline
이번 요청을 무엇으로 추적하는가	request ID·trace ID
재시도·중복을 구분할 키가 있는가	idempotency key·업무 키

4.2. correlation ID와 trace ID

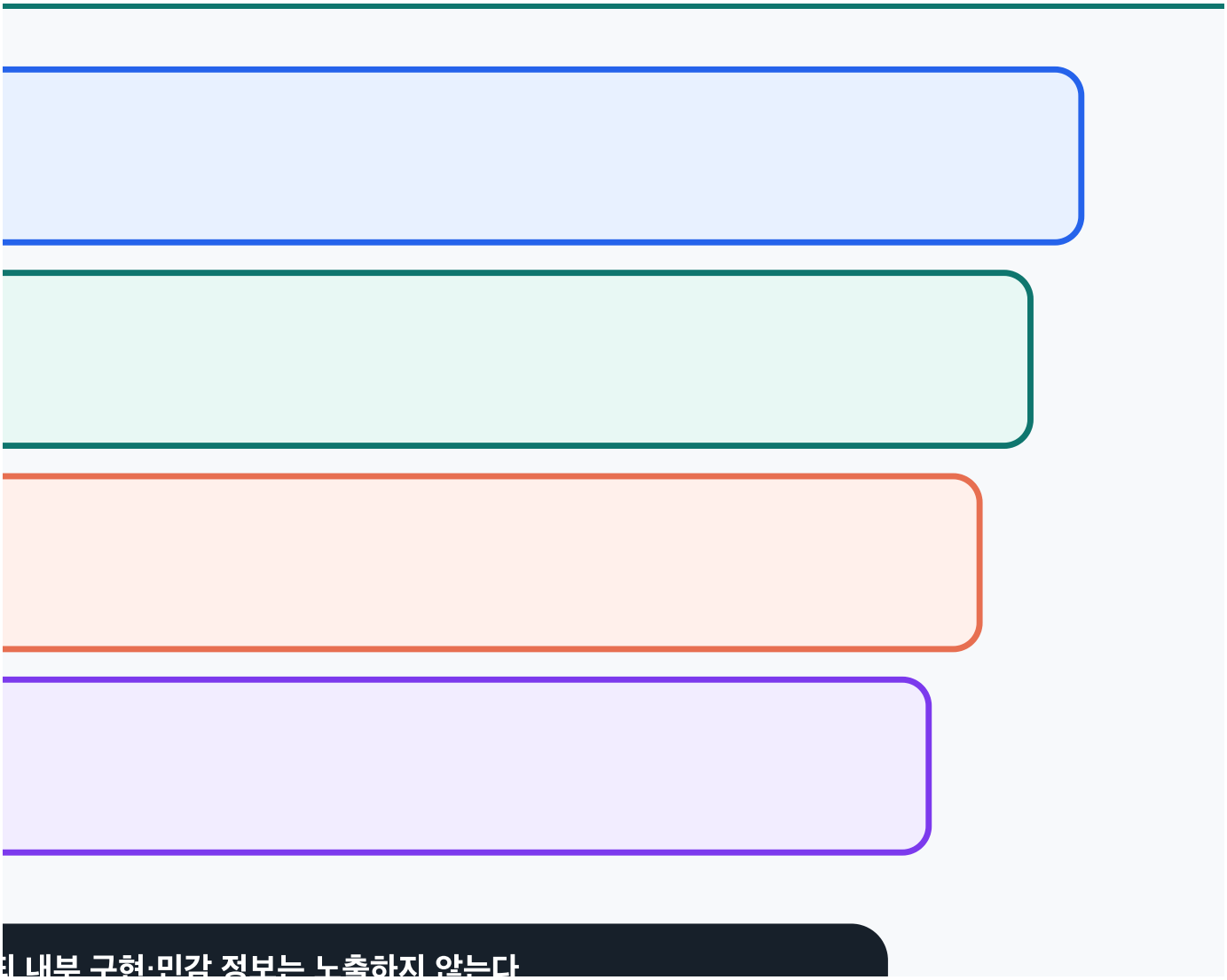
조직마다 request ID, correlation ID, trace ID 이름을 다르게 씁니다. 중요한 점은 이름보다 **경계를 지나도 같은 요청을 연결할 수 있는가**입니다. 외부에서 받은 식별자를 무조건 신뢰하지 말고 형식·길이를 제한하거나 서버가 새 값을 발급합니다.

5. 검증은 네 층으로 나누면 오류 위치가 보입니다

그림 3. JSON을 읽었다고 업무상 유효한 요청이 된 것은 아닙니다. 서버 상태와 비교하는 불변 조건까지 가야 합니다.



검증 실패는 필드와 수정 방법을 알려 주도


내부 구현·민감 정보는 노출하지 않는다

5.1. 문법 검증

요청을 파싱할 수 있는지 봅니다. 닫히지 않은 JSON, 잘못된 인코딩, 지원하지 않는 content type은 업무 규칙까지 가지 않습니다.

5.2. 구조 검증

필수 필드, 타입, 길이, 범위, 허용된 값 집합을 확인합니다.

```
productId: 비어 있지 않은 문자열
quantity: 정수, 1..10
delivery: email | download
unknown field: 거절 또는 무시 정책 명시
```

5.3. 의미 검증

각 필드가 따로 유효해도 조합은 잘못될 수 있습니다.

```
startAt < endAt
refundAmount ≤ paidAmount
delivery=email이면 emailAddress 필요
```

5.4. 불변 조건과 동시성

현재 서버 상태와 비교해야 알 수 있는 조건입니다. 재고, 잔액, 승인 상태, 이미 취소됐는지처럼 요청 사이에 바뀔 수 있는 값은 검사와 저장 사이 경쟁을 고려해야 합니다.

실패	사용자에게 줄 것	운영에 남길 것
`quantity=0`	필드·허용 범위·수정 방법	validation rule ID
종료일이 시작일 전	두 필드 관계 설명	의미 검증 결과
재고 부족	가능한 다음 행동	현재 재고·rule decision
허용하지 않은 필드	일반적인 요청 오류	필드명·정책, 민감값 제외

30초 확인

`quantity`가 정수 5이지만 재고가 4라면 구조 검증은 통과하고 불변 조건에서 실패합니다. 두 실패를 한꺼번에 “validation error”로 부르면 사용자가 무엇을 고칠지, 운영자가 어느 규칙을 봐야 할지 흐려집니다.

6. 인증·인가·업무 규칙을 구분합니다

그림 4. 로그인 여부, 객체·행동 권한, 현재 업무 조건은 연속해서 물을 수 있지만 판정 근거와 거절 이유가 다릅니다.

AUTHENTICATION

인증

당신은 누구인가

대표 판정

token 만료 → 401

판정 주체와 근거 데이터를 분리

모든 객체·행동에서 다시 확인

AUTHORIZATION

인가

이 주문을 수정할 수 있는가

대표 판정

타인 주문 → 403

판정 주체와 근거 데이터를 분

모든 객체·행동에서 다시 확인

로그인 성공은 모든 데이터 접근 권한을 뜻하지 않

BUSINESS RULE

업무 규칙

지금 취소할 수 있는가

대표 판정

배송 시작 → 409

판정 주체와 근거 데이터를 분리

모든 객체·행동에서 다시 확인

고, 권한 보유는 업무 시점 허용을 뜻하지 않는다

6.1. 인증 · 누구인가

세션·token·인증서 등으로 principal을 확인합니다. 인증 정보가 없거나 유효하지 않으면 보통 401 범주의 계약을 검토합니다. 구체적인 인증 설계는 M04-03에서 다룹니다.

6.2. 인가 · 이 객체에 이 행동을 할 수 있는가

OWASP API Security Top 10 2023은 클라이언트가 보낸 객체 ID로 데이터에 접근하는 모든 기능에서 객체 수준 권한을 확인하도록 안내합니다. 로그인한 사용자라도 타인의 주문 ID를 바꾸어 보내면 거절되어야 합니다.

6.3. 업무 규칙 · 권한이 있어도 지금 가능한가

주문 소유자가 취소 권한을 갖고 있어도 배송이 시작된 뒤에는 취소가 불가능할 수 있습니다. 이것은 정체성 실패가 아니라 현재 주문 상태와 정책의 충돌입니다.

질문	판정 근거	예시 결과
누구인가	session·token 검증	anonymous·user:1024
이 행동을 할 수 있는가	role·scope·소유권·정책	allow·deny
지금 허용되는가	상태·기간·재고·업무 정책	cancelable·conflict

안전한 ID만으로는 권한이 생기지 않습니다.
순차 숫자 대신 UUID를 써도 해당 객체를 볼 권한을 서버에서 확인해야 합니다. 예측하기 어려운 식별자는 보조 수단이지 접근 제어가 아닙니다.

7. 트랜잭션으로 함께 성공할 변경을 묶습니다

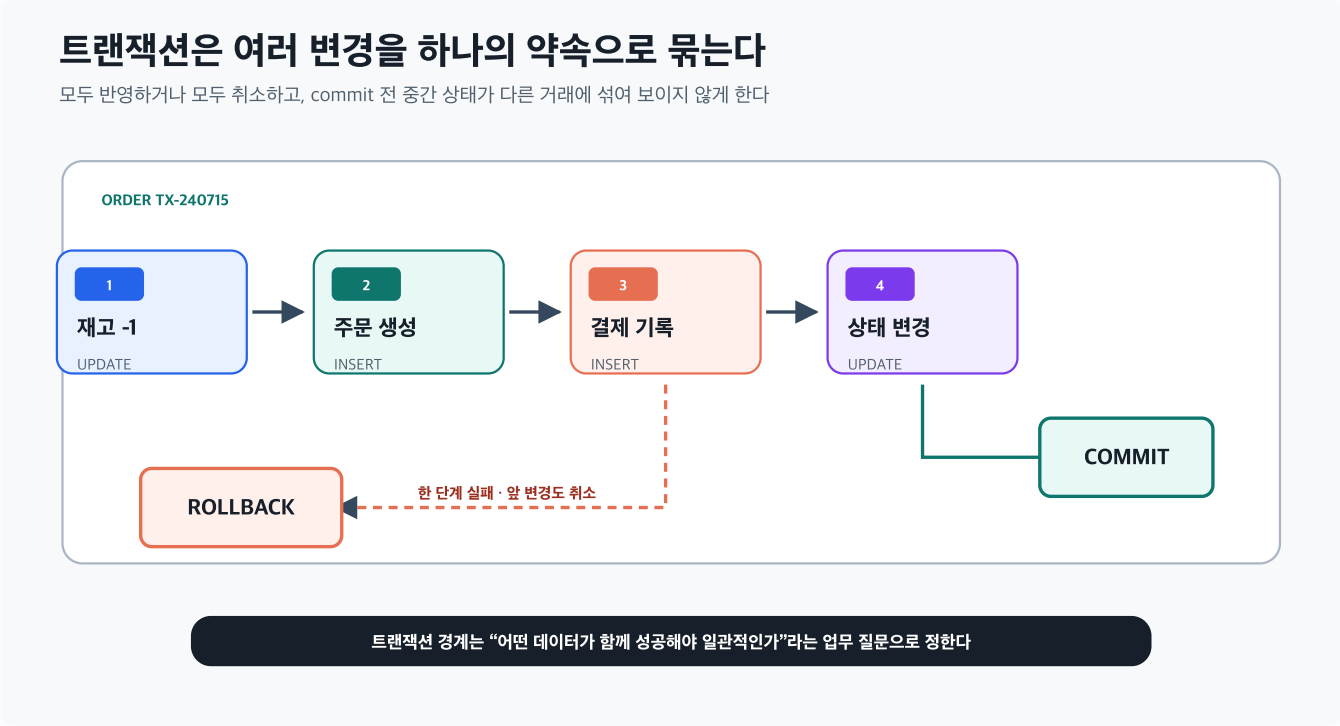
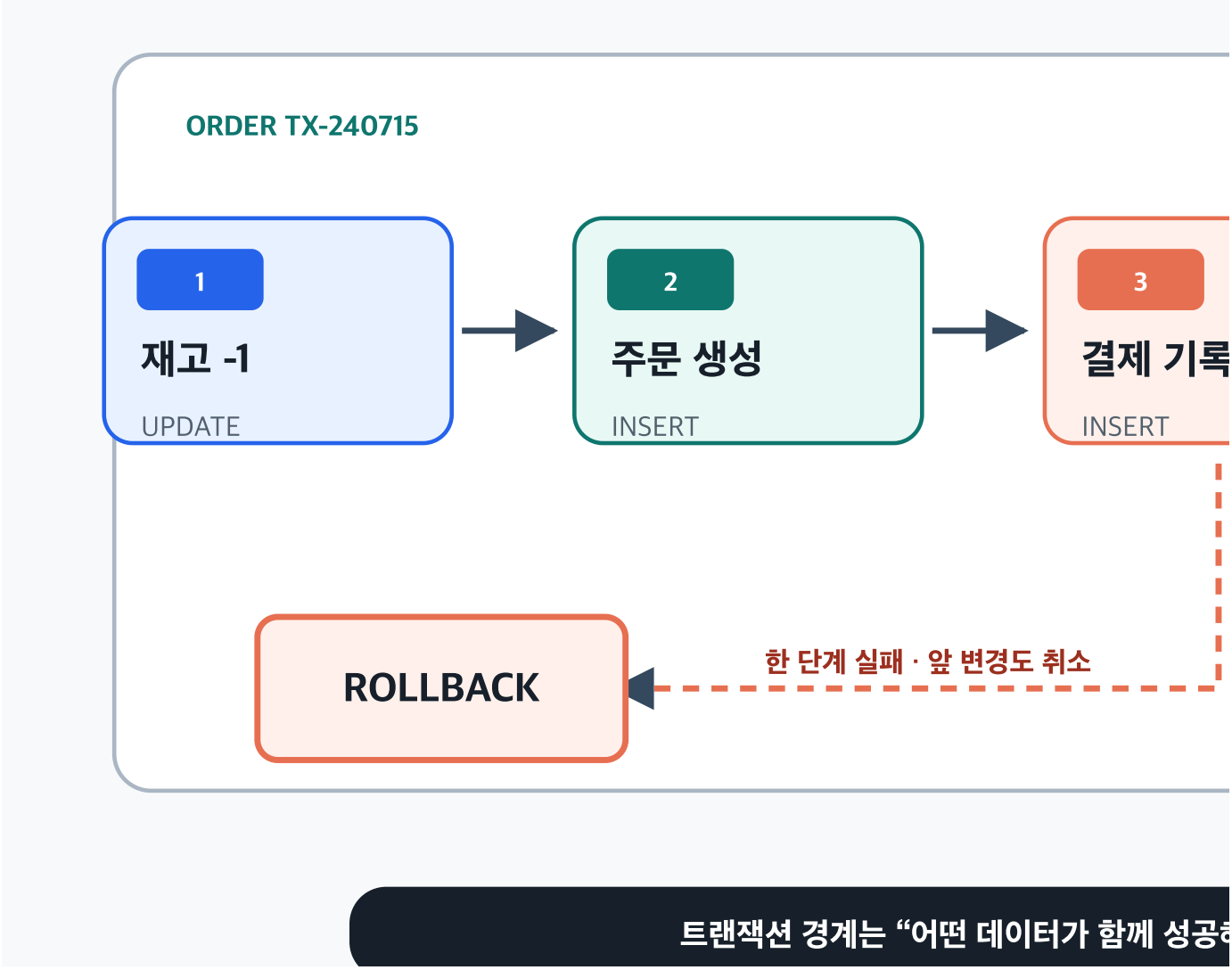
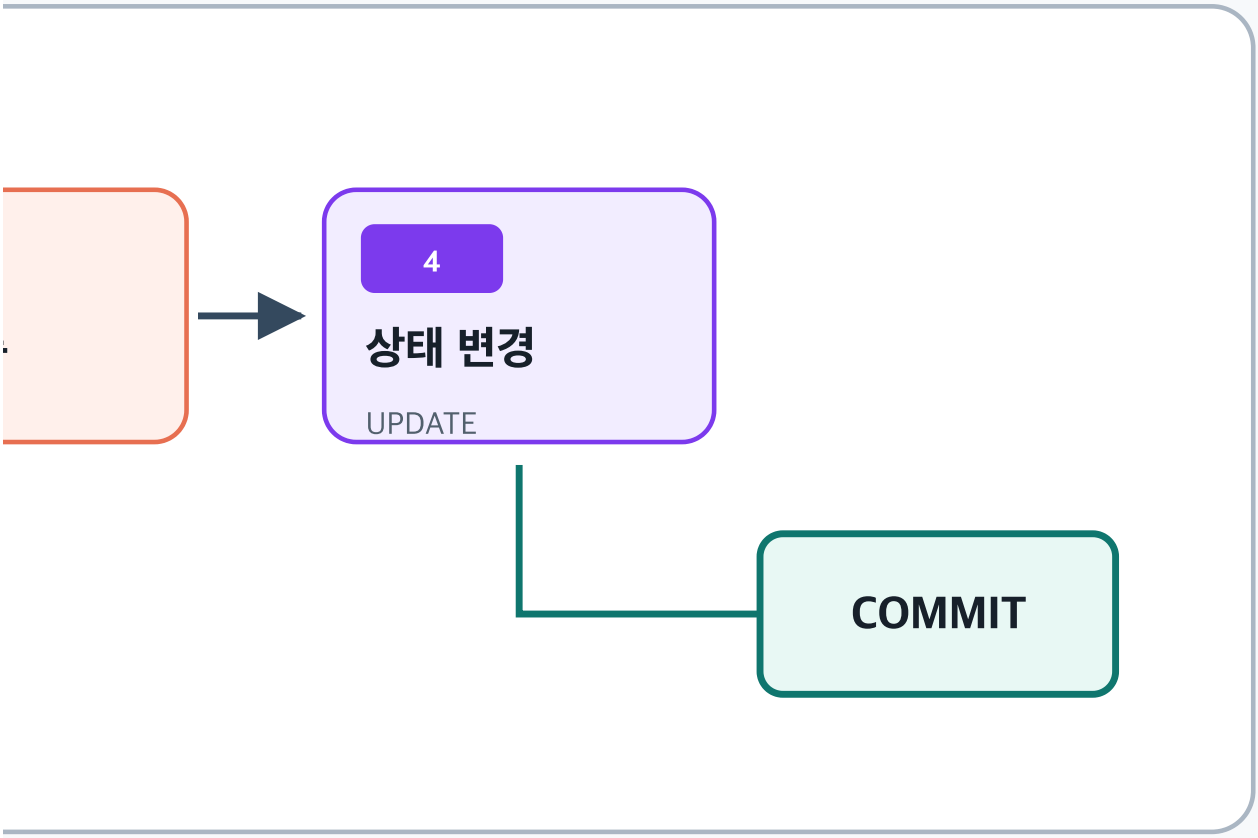


그림 5. 재고 감소·주문 생성·결제 기록·상태 변경이 하나의 일관된 결과라면 중간 실패 때 앞 변경도 되돌아가야 합니다.


ORDER TX-240715
1
재고 -1
UPDATE
2
주문 생성
INSERT
3
결제 기록
INSERT
ROLLBACK
한 단계 실패 · 앞 변경도 취소
트랜잭션 경계는 “어떤 데이터가 함께 성공



해야 일관적인가”라는 업무 질문으로 정한다

PostgreSQL 공식 문서는 트랜잭션을 여러 단계를 하나의 all-or-nothing operation으로 묶는 개념으로 설명합니다. 열린 트랜잭션의 변경은 완료될 때까지 다른 트랜잭션에 보이지 않고, 완료되면 함께 보입니다.

7.1. begin·commit·rollback

```
BEGIN
  stock 4 → 3
  order 12 → 13
  payment record insert
COMMIT
```

세 번째 단계에서 실패하면 `ROLLBACK` 뒤 재고는 4, 주문 수는 12여야 합니다. “DB 오류가 났지만 앞의 UPDATE는 남았다”면 거래의 업무 의미가 깨집니다.

7.2. 트랜잭션 경계 질문

질문	판단 예
무엇이 함께 성공해야 일관적인가	주문과 재고 예약
무엇은 나중에 해도 되는가	영수증 이메일
외부 호출을 열린 DB tx 안에서 기다려야 하는가	lock·지연·불확실성 검토
commit 뒤 이벤트 발행 실패는 어떻게 복구할까	outbox·재처리 등 후속 교재에서 확장
재시도 시 같은 변경이 반복되는가	업무 키·idempotency 설계

7.3. 저장 성공 증거

`commit` 로그 한 줄만 보지 않습니다. record ID, 새 상태, transaction 결과, 기대한 행 수, 사용자에게 반환한 resource location을 연결합니다.

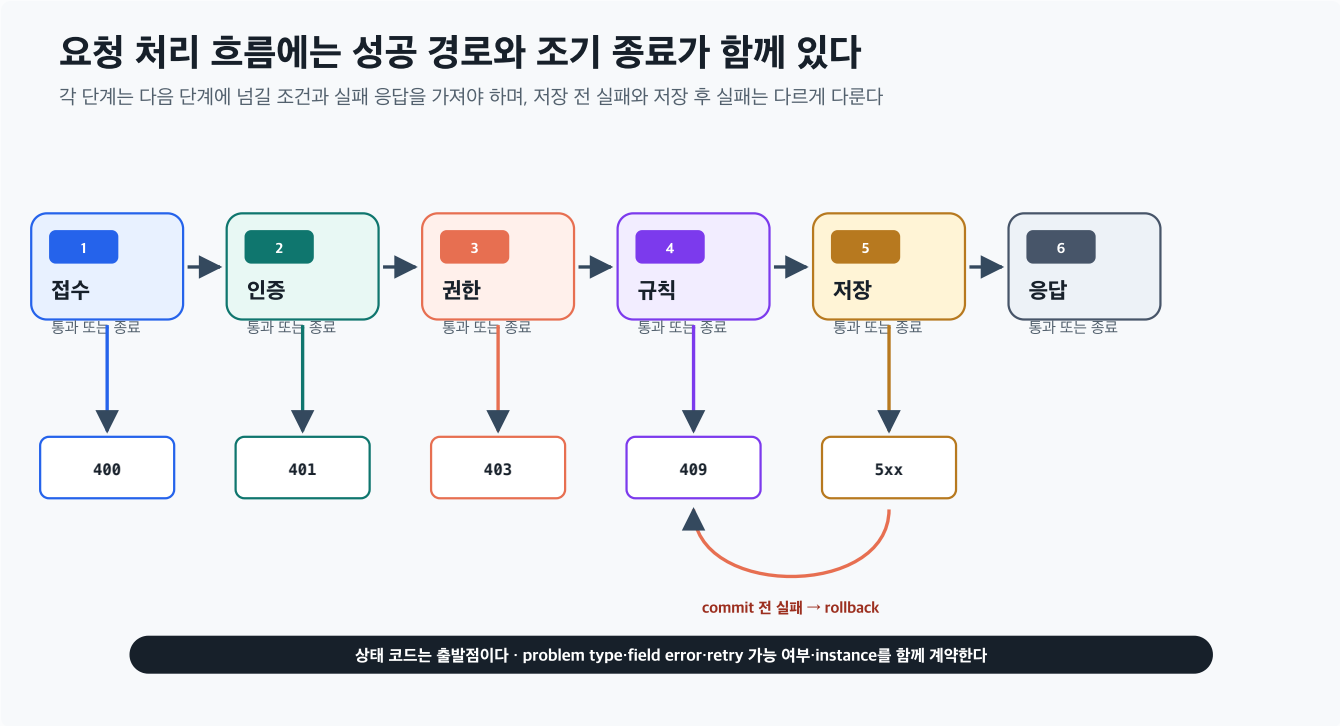
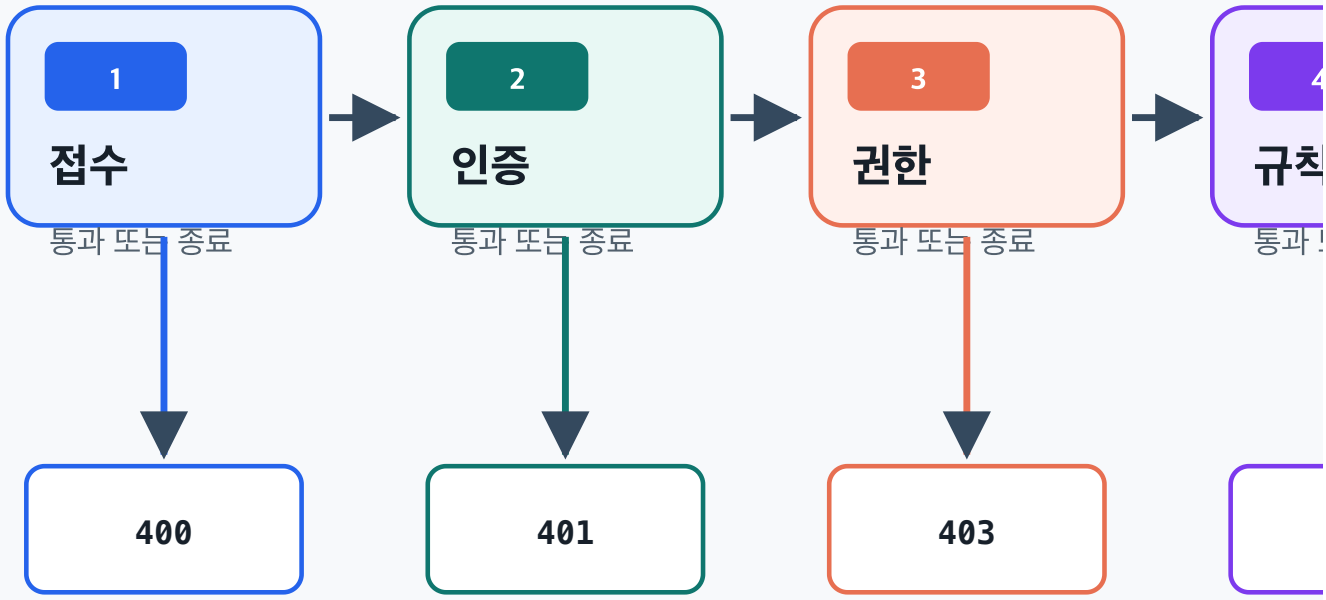
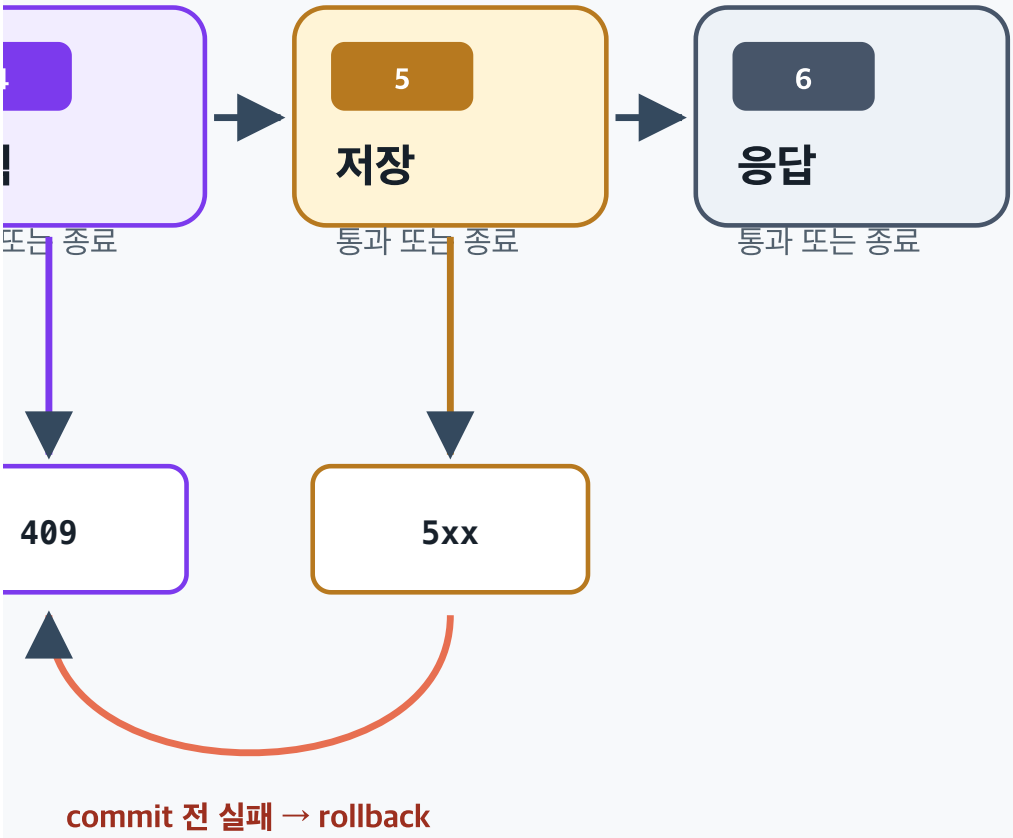


그림 6. 실패한 문 뒤의 단계는 실행하지 않습니다. 저장 전 실패와 transaction 중 실패의 데이터 결과가 같은지도 검수합니다.



상태 코드는 출발점이다 · problem type·field e



error·retry 가능 여부·instance를 함께 계약한다

8. 외부 연동에는 실패 예산을 적습니다

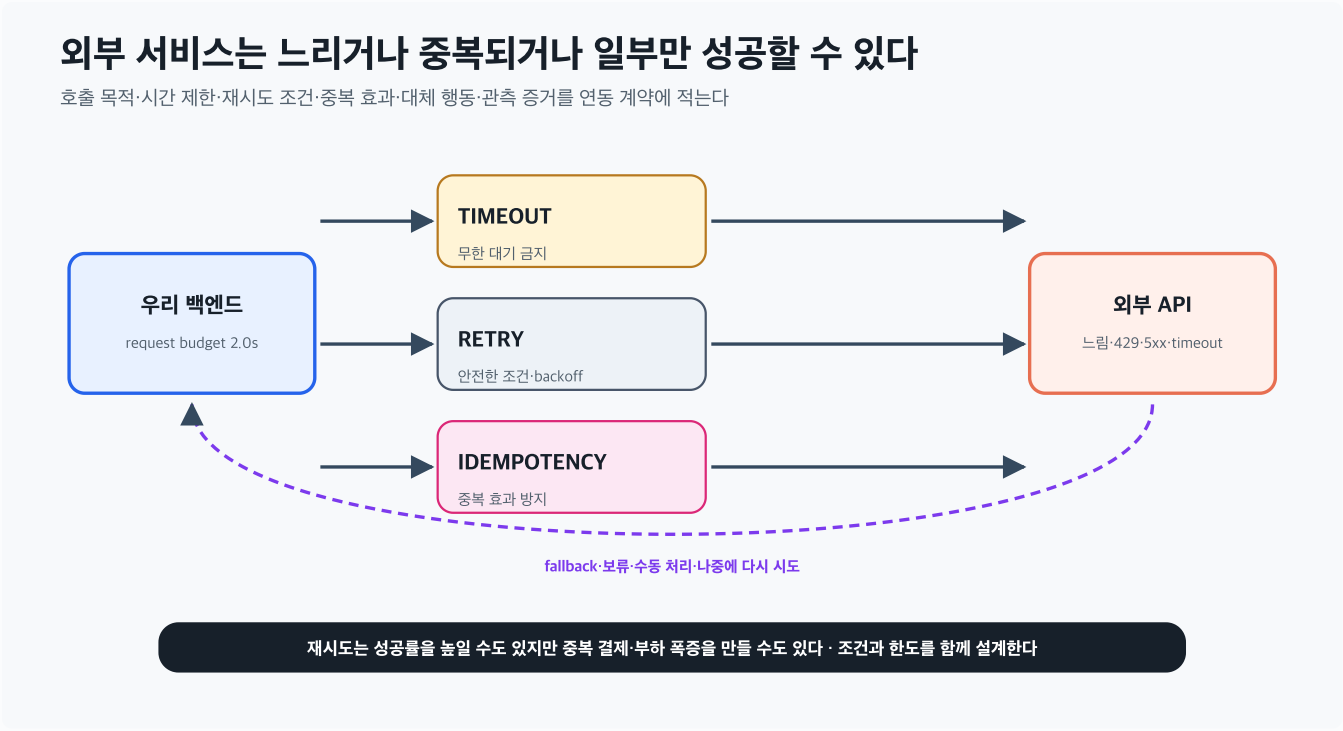
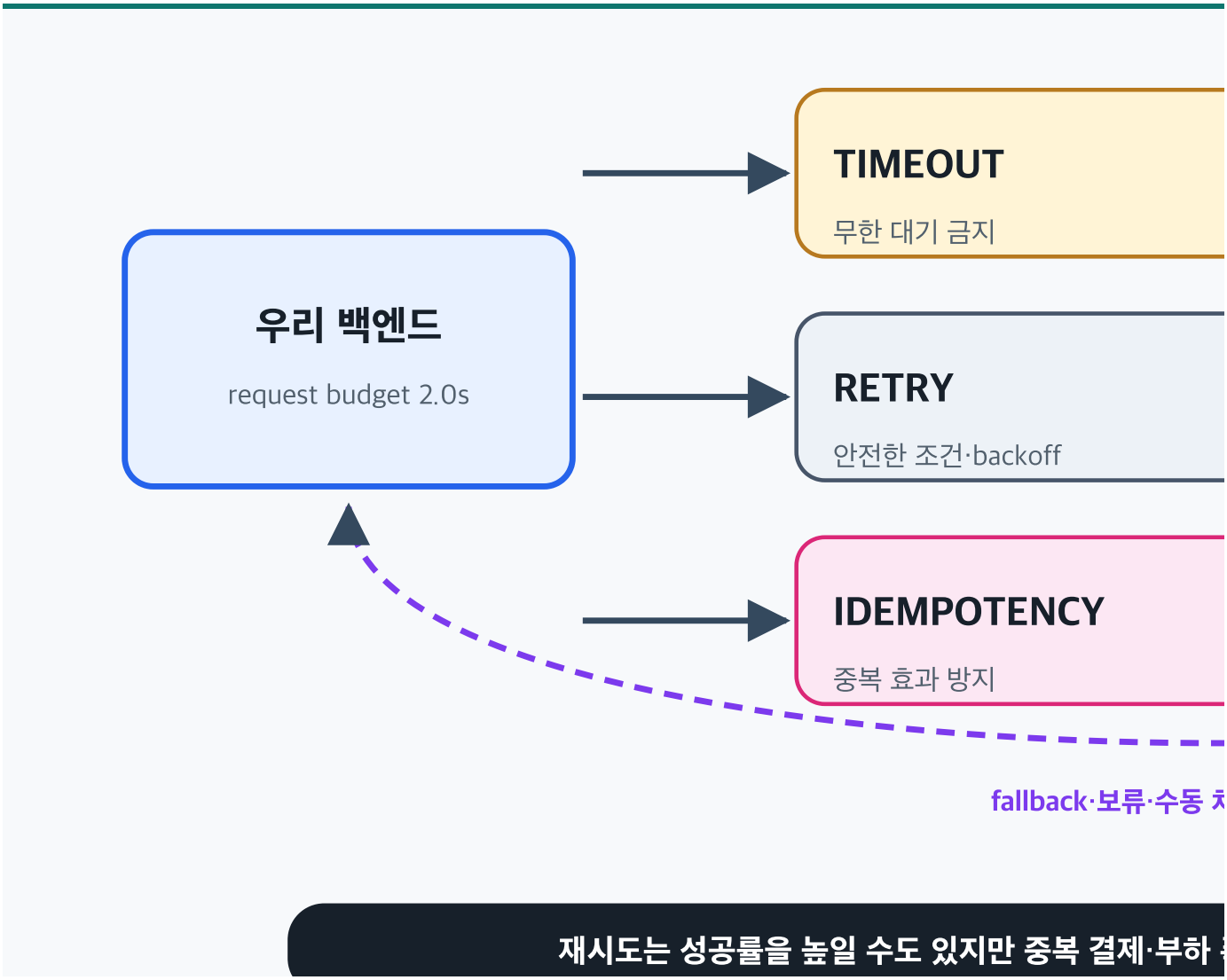
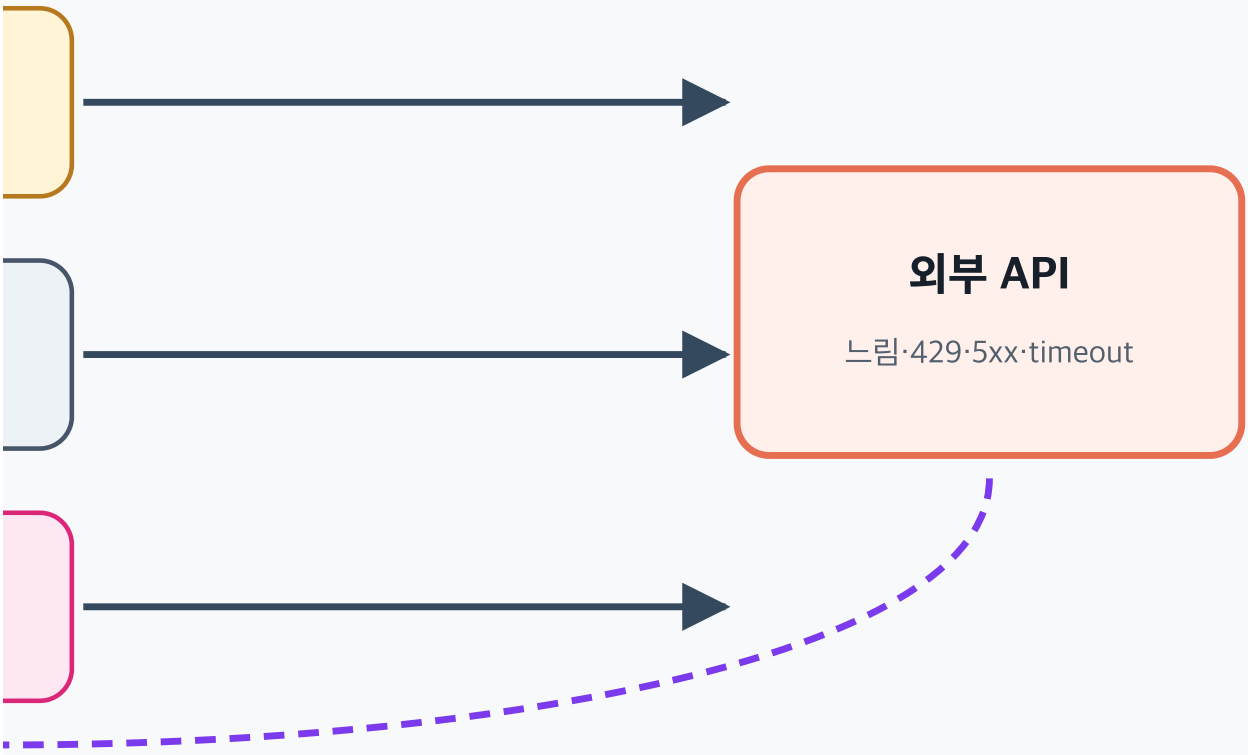


그림 7. 외부 호출은 성공 응답 예시보다 timeout·재시도·중복 효과·대체 행동을 먼저 적으면 운영 위험이 보입니다.


우리 백엔드
request budget 2.0s
TIMEOUT
무한 대기 금지
RETRY
안전한 조건·backoff
IDEMPOTENCY
중복 효과 방지
fallback·보류·수동 처
재시도는 성공률을 높일 수도 있지만 중복 결제·부하



폭증을 만들 수도 있다 · 조건과 한도를 함께 설계한다

외부 API·메일·문자·결제·AI 모델은 우리 트랜잭션과 다른 실패 경계를 가집니다. 응답이 늦다고 실패했는지, 성공했지만 응답만 잃었는지 모를 수 있습니다.

8.1. 연동 계약의 여섯 질문

1. 무엇을 얻거나 바꾸기 위한 호출인가?
2. 전체 요청 시간 중 얼마나 기다릴 수 있는가?
3. 어떤 오류와 method를 다시 시도해도 되는가?
4. 같은 요청이 두 번 도착하면 효과가 두 번 생기는가?
5. 실패하면 rollback·보류·대체·수동 처리 중 무엇을 하는가?
6. attempt와 외부 request ID를 어디에 남기는가?

8.2. retry는 기본 정답이 아닙니다

RFC 9110은 safe·idempotent method 의미를 구분합니다. 같은 의도 효과를 반복해도 한 번과 같다는 속성이 확인되지 않은 변경 요청을 자동 재시도하면 중복 결제·중복 신청을 만들 수 있습니다.

상황	바로 retry?	먼저 확인할 것
연결 전 실패	조건부 가능	요청이 실제 전송됐는가
429	서버 지시에 따라	`Retry-After`, 전체 한도
5xx	제한적으로	method 효과·backoff·최대 시도
timeout	불확실	외부 결과 조회·idempotency key
4xx 입력 오류	보통 아니요	요청 수정 가능성

9. 동기와 비동기의 완료 경계를 계약합니다

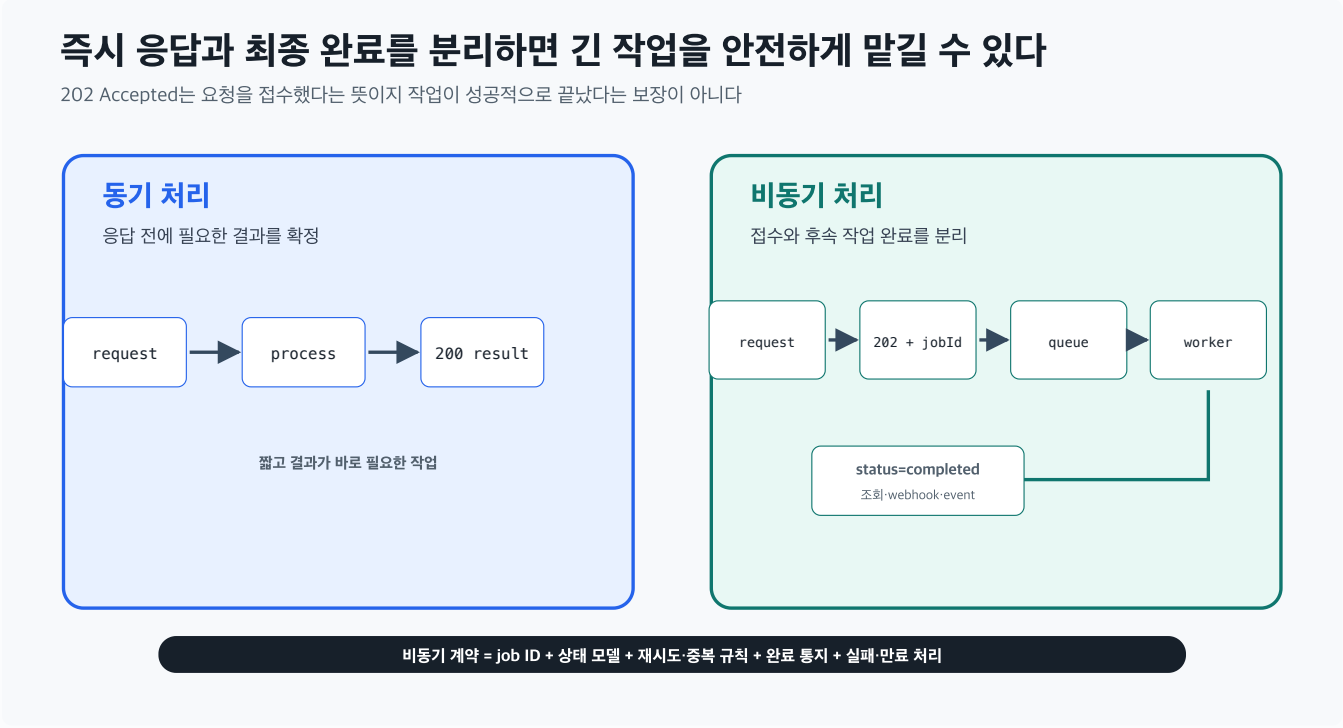


그림 8. 202는 접수 응답입니다. 최종 결과는 job 상태 조회·webhook·event 같은 별도 통로로 알려야 합니다.

동기 처리

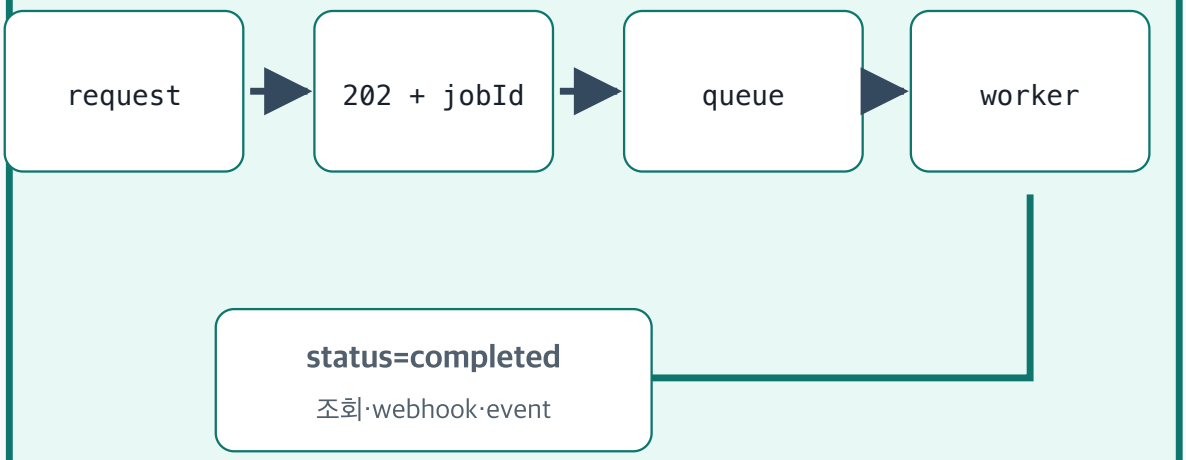
응답 전에 필요한 결과를 확정

짧고 결과가 바로 필요한 작업

비동기 계약 = job ID + 상태 모델 + 재시

비동기 처리

접수와 후속 작업 완료를 분리



도·중복 규칙 + 완료 통지 + 실패·만료 처리

9.1. 동기 처리

응답을 보내기 전에 계약한 결과를 확정합니다. 짧고 즉시 결과가 필요한 조회·변경에 적합하지만, 긴 외부 작업을 계속 기다리면 timeout과 자원 점유가 커집니다.

9.2. 비동기 처리

요청을 접수하고 job이나 event를 남긴 뒤 먼저 응답합니다. worker가 나중에 처리하며 실패·재시도·완료 통지가 별도입니다.

RFC 9110의 `202 Accepted`는 처리를 받아들였지만 완료되지 않았고, 나중에 실행되지 않을 수도 있는 비확정적 응답입니다. 응답 표현은 현재 상태와 상태를 확인할 방법을 제공하는 것이 좋습니다.

```
{
  "jobId": "JOB-81",
  "state": "queued",
  "statusUrl": "/jobs/JOB-81",
  "submittedAt": "2026-07-15T09:20:00+09:00"
}
```

9.3. 비동기 상태 모델

```
queued → running → completed
          ├→ retrying → running
          ├→ failed
          └→ canceled
```

항목	반드시 정할 것
식별	job ID·업무 대상 ID
상태	허용 상태와 전이
중복	같은 event·job 재전달 처리
재시도	횟수·간격·최종 실패
완료	결과 위치·완료 시각·사용자 통지
만료	결과 보존 기간·재실행 방법

CloudEvents는 서로 다른 서비스가 event를 공통 형식으로 주고받기 위한 vendor-neutral 규격입니다. `id`, `source`, `specversion`, `type` 같은 문맥 속성과 domain data를 구분합니다. 다만 queue의 처리 보장이나 우리 업무의 재시도 정책까지 대신 정해 주지는 않습니다.

30초 확인

사용자가 202를 받았다는 사실과 보고서가 완성됐다는 사실 사이에는 worker 실행·실패·재시도·저장·통지 단계가 남아 있습니다. 화면도 `접수 완료`와 `최종 완료`를 같은 문구로 표시하면 안 됩니다.

10. 오류를 인터페이스 계약으로 만듭니다

그림 9. status는 큰 의미, type은 안정적인 오류 종류, instance는 이번 발생, detail은 이번 건의 수정 설명을 담당합니다.

HTTP/1.1 409 Conflict

```
"type":      "https://yeoncore.ai/problems/stock"

"title":     "재고가 부족합니다"

"status":    409

"detail":    "수량을 3 이하로 바꾸세요"

"instance":  "/problems/prb-8f31"

"errors":    [{"field":"quantity"}]
```

status

HTTP의 큰 의미

type

안정적인 오류 종류

detail

이번 건의 수정 설명

instance

이번 발생의 추적 ID

extension

field·retry 등 계약

stack trace·SQL·secret은 응답에 넣지 않는다

RFC 9457은 HTTP API 오류를 기계가 읽을 수 있게 표현하는 `application/problem+json` 형식을 정의합니다. 기본 멤버는 `type`, `status`, `title`, `detail`, `instance`이며 필요하면 확장 멤버를 더할 수 있습니다.

```
{
  "type": "https://yeoncore.ai/problems/stock-conflict",
  "title": "재고가 부족합니다",
  "status": 409,
  "detail": "수량을 3 이하로 바꾸세요.",
  "instance": "/problems/prb-8f31",
  "errors": [{"field": "quantity", "max": 3}]
}
```

10.1. 멤버의 역할

멤버	안정성	용도
`type`	같은 오류 종류에 안정적	클라이언트 분기·문서 연결
`title`	종류별 짧은 요약	사람이 빠르게 이해
`status`	HTTP 응답과 일치	일반 HTTP 처리
`detail`	발생 건마다 달라짐	사용자가 고치도록 설명
`instance`	이번 발생 식별	문의·운영 추적
확장	계약으로 정의	field errors·retry 등 구조화 정보

`detail` 문자열을 클라이언트가 파싱해 분기하지 않습니다. 기계 판정은 안정적인 `type`과 구조화 확장 필드를 씁니다. stack trace·SQL·내부 host·secret·권한 정책 상세는 사용자 응답에 노출하지 않습니다.

10.2. 오류 분류표

오류 질문	예시	저장 결과	사용자 다음 행동
요청을 읽을 수 있나	malformed JSON	없음	요청 수정
인증됐나	session expired	없음	다시 로그인
이 객체를 다룰 수 있나	타인 주문	없음	권한 있는 계정·문의
현재 업무상 가능한가	재고 부족	없음	수량 변경
저장됐나	DB unavailable	rollback 확인	안전하게 재시도

오류 질문	예시	저장 결과	사용자 다음 행동
후속 작업이 끝났나	job failed	접수 record 존재	상태 확인·재실행

11. 관측 증거로 같은 요청을 다시 만납니다

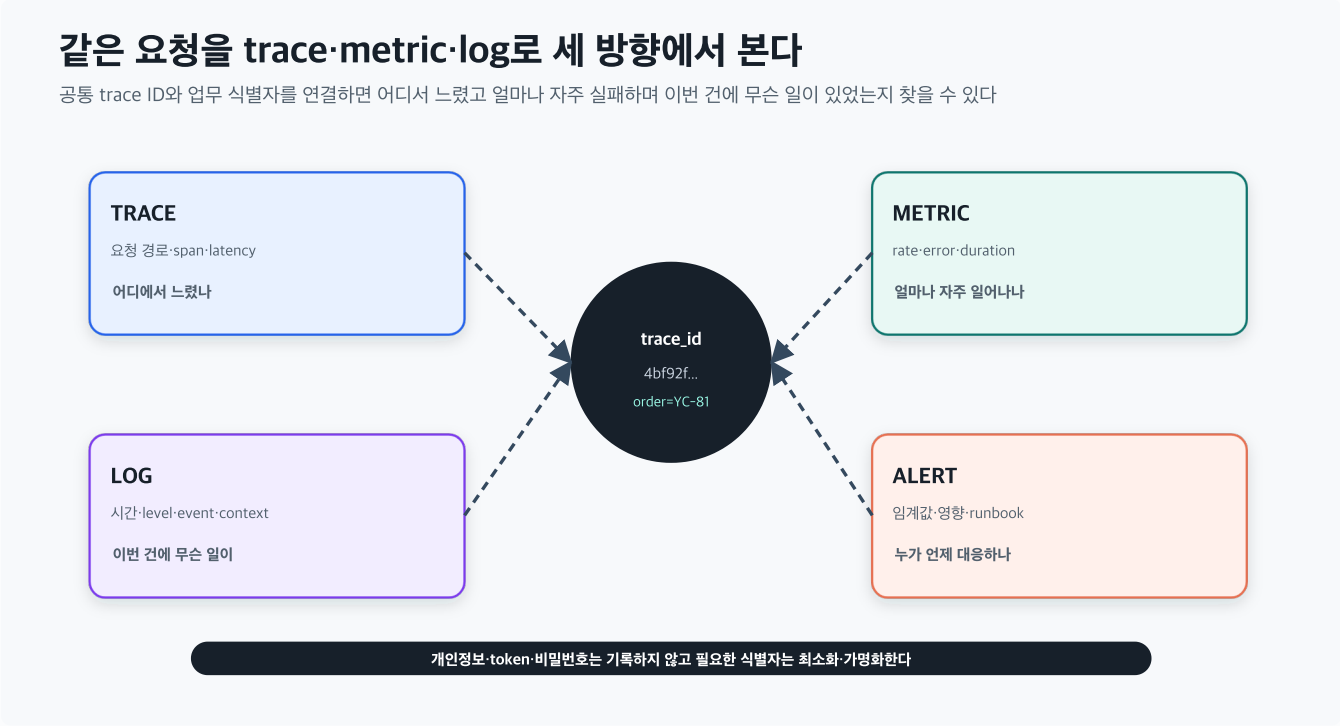
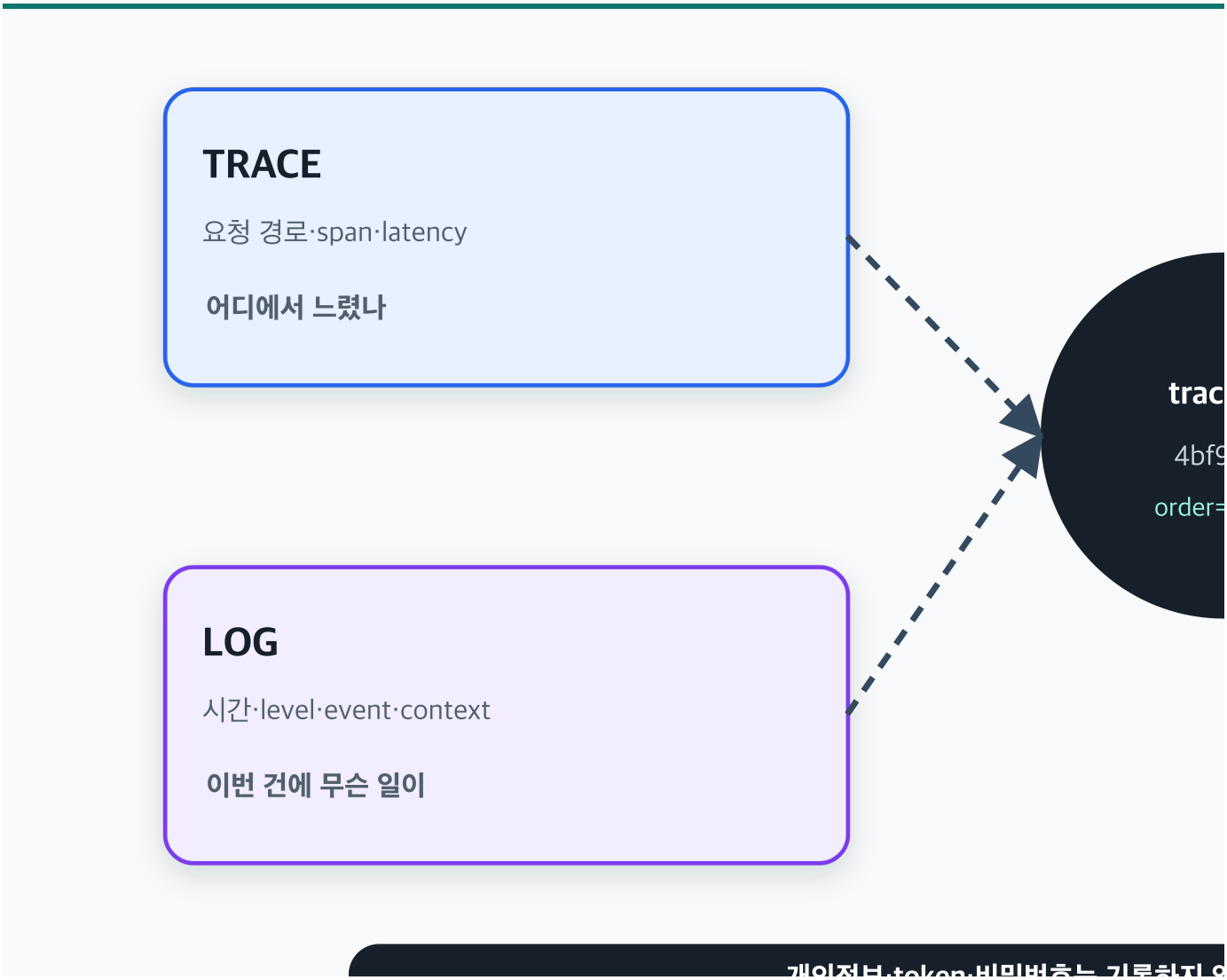
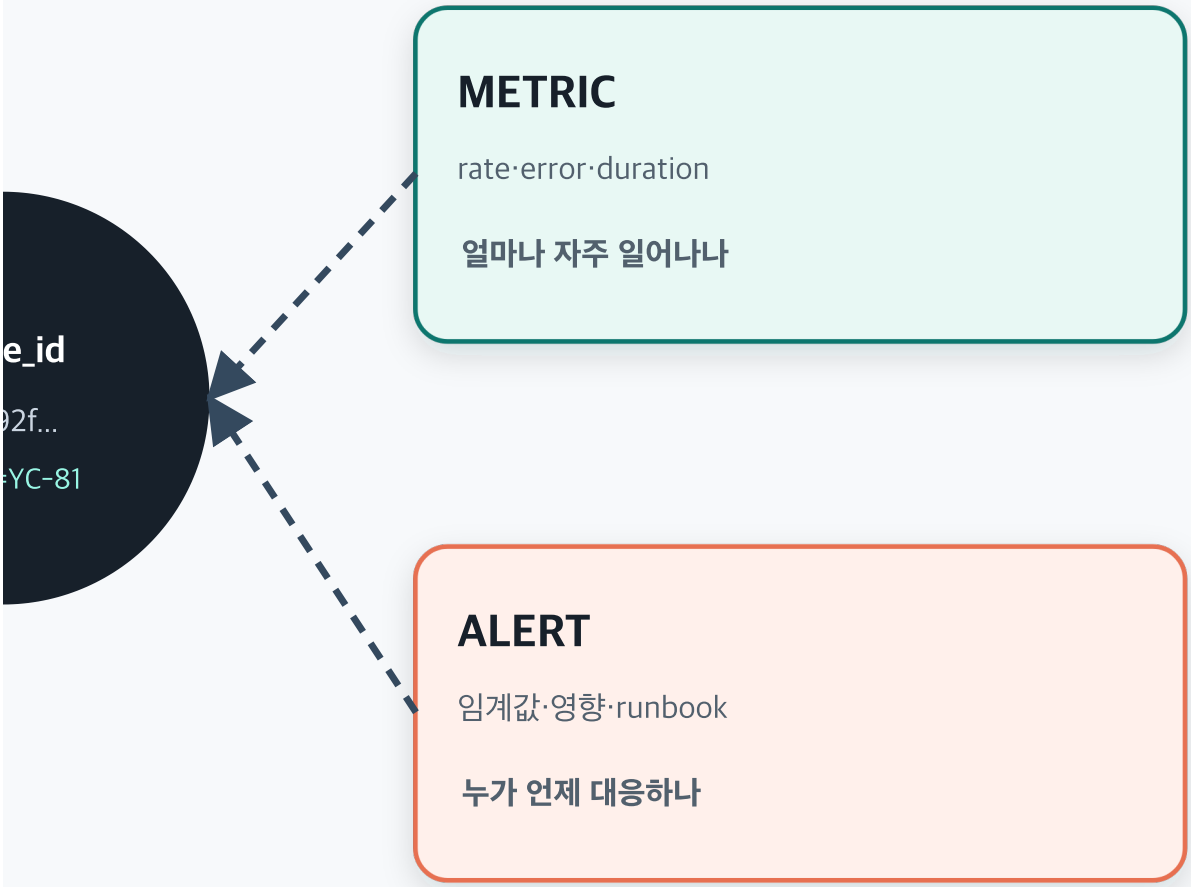


그림 10. trace·metric·log는 서로 대신하지 않습니다. 같은 문맥 ID로 연결할 때 경로·빈도·이번 발생을 함께 볼 수 있습니다.


TRACE
요청 경로·span·latency
어디에서 느렸나
LOG
시간·level·event·context
이번 건에 무슨 일이
trac
4bf9
order=


METRIC
rate·error·duration
얼마나 자주 일어나나
ALERT
임계값·영향·runbook
누가 언제 대응하나

OpenTelemetry는 trace·metric·log를 관측 신호로 다루며 context propagation으로 서비스 경계의 신호를 연관 지을 수 있게 합니다. Trace ID와 Span ID를 log record에 넣어 같은 요청을 연결할 수 있습니다.

11.1. 세 신호가 답하는 질문

신호	질문	주문 예
trace	어디를 거쳐 어디서 느렸나	API 80ms → payment 1.6s
metric	얼마나 자주·얼마나 심한가	5분 error rate 4.2%
log	이번 건에 무슨 일이 있었나	rule=stock, decision=deny

11.2. 구조화 로그 최소 필드

```
{
  "timestamp": "2026-07-15T09:20:03.221+09:00",
  "level": "WARN",
  "event": "order.rule.denied",
  "trace_id": "4bf92f...",
  "order_id": null,
  "rule": "stock_available",
  "decision": "deny",
  "reason_code": "STOCK_CONFLICT"
}
```

메시지 한 줄에 모든 것을 넣기보다 검색·집계할 필드를 구조화합니다. 비밀번호, token 원문, 주민등록번호, 전체 요청 본문 등 민감 정보는 기록하지 않습니다.

11.3. 성공도 관측합니다

오류만 기록하면 분모가 없습니다. 요청 수, 성공 수, latency 분포, rollback 수, 외부 attempt 수, queue 대기 시간, job 완료율을 함께 봅니다.

12. 프런트·계약·백엔드 책임을 표로 맞춥니다

그림 11. 양쪽이 같은 항목을 다뤄도 프런트는 사용자 경험, 백엔드는 권위 있는 판정, API 계약은 둘의 합의를 담당합니다.

프런트

사용 경험·표현

- 입력 도움
- loading·error 화면
- 중복 클릭 방지
- 접근 가능한 피드백

API 계약

요청·응답·오류

- method·path·schema
- 성공·오류 status
- problem type·field error
- job status·retry 조건
- trace·business ID

책임표의 핵심 열 = 판정 주체 · 입력 근

백엔드

권위 있는 판정·변경

- 입력 재검증
- 인증·객체 권한
- 업무 규칙
- transaction·storage

근거 · 성공 증거 · 실패 복구 · 관측 ID

관심사	프런트 책임	API 계약	백엔드 책임
입력	즉시 피드백	request schema	모든 입력 재검증
권한	불가능 행동 표현	auth requirement·403	객체·행동 최종 판정
중복	버튼 잠금·진행	key·replay semantics	중복 효과 방지
저장	loading·완료 화면	201·Location·resource	transaction·commit
비동기	접수·진행·완료 구분	202·job schema	queue·worker·retry
오류	수정 가능한 화면	problem type·fields	내부 오류 매핑·민감 정보 보호
추적	참조 번호 표시	instance·request ID	trace·log·record 연결

12.1. “공동 책임”을 빈칸으로 쓰지 않습니다

“둘 다”라고만 쓰면 최종 판정 주체가 사라집니다. 각 행에 `표시`, `계약`, `권위 있는 결정`, `저장`, `운영 대응`을 구분합니다.

12.2. OpenAPI가 돕는 부분

OpenAPI Specification은 HTTP API의 paths, operations, parameters, request body, responses, security 등을 기계가 읽을 수 있게 기술합니다. 그러나 문서에 schema를 적었다고 runtime 검증·권한 검사·업무 규칙·transaction이 자동으로 올바르게 구현되는 것은 아닙니다.

13. 실습 · 아홉 경로의 증거를 비교합니다

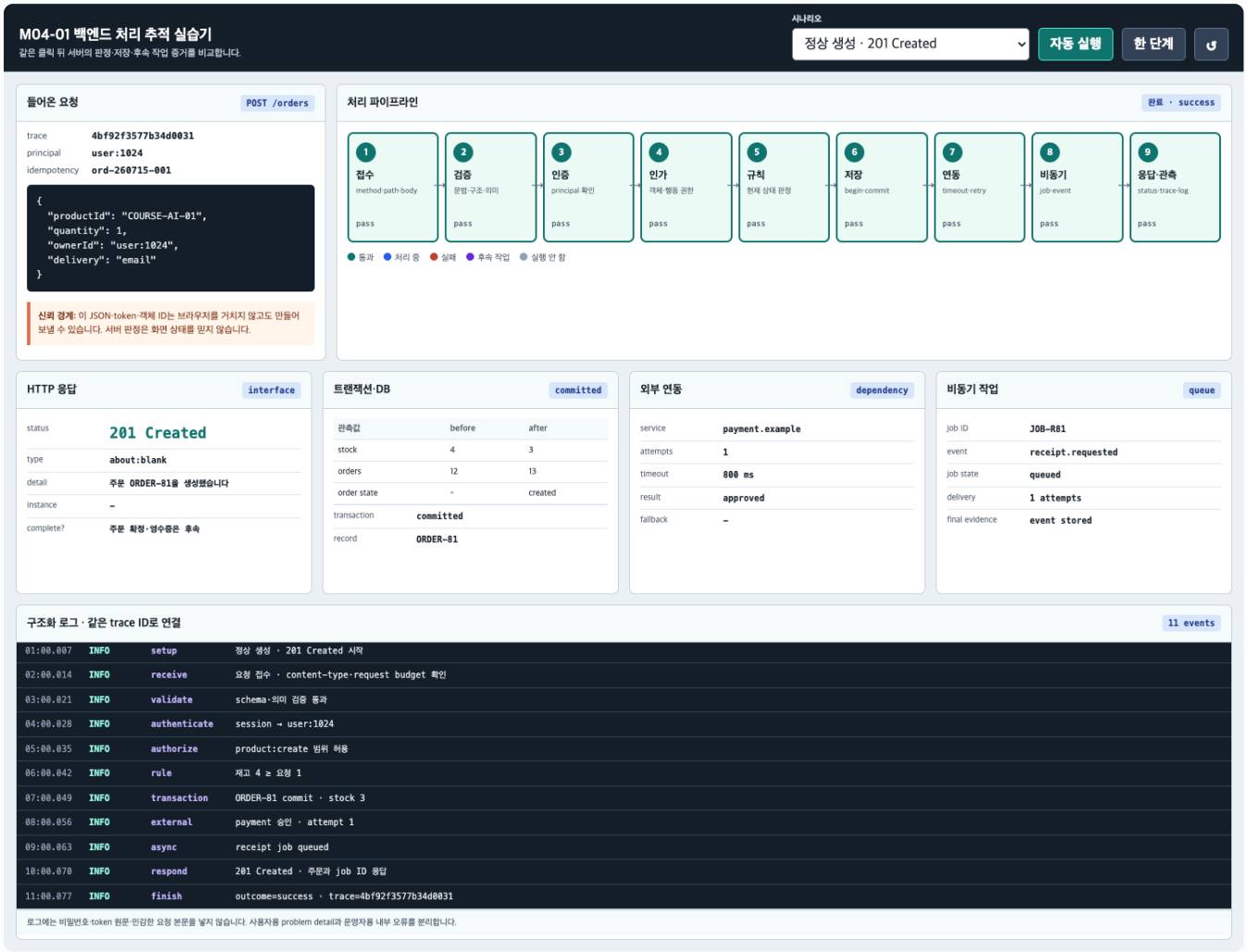


그림 12. 실습기는 같은 화면 동작을 응답·DB 전후·외부 attempt·job 상태·구조화 로그로 동시에 보여 줍니다.

13.1. 준비 파일

- 실습 안내
- 백엔드 처리 추적 실습기
- 백엔드 처리 지도 템플릿
- 백엔드 처리 용어집

13.2. 먼저 예측하고 실행합니다

시나리오	멈출 문	예상 응답	DB after	후속 작업
정상 생성	응답	201	stock 3·orders 13	영수증 job
구조 검증 실패	검증	400	변화 없음	없음
인증 실패	인증	401	변화 없음	없음
객체 권한 실패	인가	403	변화 없음	없음
업무 규칙 충돌	규칙	409	변화 없음	없음
DB 저장 실패	저장	503	rollback	없음
외부 timeout	연동	503	rollback	attempt 2
비동기 접수	응답 뒤 worker	202	job record	completed event
중복 요청	접수·응답	기존 결과	한 번만 변경	한 번만 생성

13.3. 반드시 적을 네 증거

1. 사용자에게 보인 HTTP status·problem detail
2. transaction과 DB before·after
3. 외부 attempt 또는 job·event 상태
4. 같은 건을 연결하는 trace ID·record ID·instance

실습 통과 기준
아홉 시나리오 중 여섯 개 이상에서 “어디서 멈췄다”뿐 아니라 “뒤 단계는 왜 실행되지 않았고 데이터와 후속 작업이 어떤 상태로 남았는지”를 설명합니다.

14. 실무 예제 · 교육 신청 승인

14.1. 약한 요구

```
신청서를 저장하고 담당자에게 메일을 보낸다.
```

14.2. 처리 지도

```
POST /applications
  → JSON·필수 필드 검증
  → user:1024 인증
  → 본인 신청 생성 권한
  → 모집 기간·중복 신청·정원 규칙
  → application + seat reservation commit
  → notification.requested event 기록
  → 201 + applicationId + notificationJobId
```

14.3. 실패 계약

실패	응답	데이터	복구
필수 증빙 없음	400 field error	없음	증빙 추가
모집 종료	409 period-closed	없음	다음 과정 안내
같은 과정 중복	409 duplicate	기존 신청 유지	기존 신청 열기
DB 실패	503 storage-unavailable	rollback	key 유지해 재시도
메일 worker 실패	201 유지·job failed	신청 존재	운영 재처리·화면 상태 안내

메일이 실패했다고 신청을 없앨지, 신청은 유지하고 알림만 재처리할지는 업무 결정입니다. 기술 편의로 정하지 말고 사용자에게 약속한 완료 경계를 기준으로 정합니다.

그림 13. 한 장 요약. 각 문마다 ID·판정·상태 변경·후속 작업·응답 중 적어도 하나의 증거가 남습니다.

1

접수

request ID

2

검증

field error

5

저장

commit·record

6

연동

timeout·attempt

요청 증거 · 상태 변경 증거 · 후속

완료 기준 = 성공뿐 아니라 거절·실패·중복

3

권한

principal·scope

4

규칙

decision reason

7

비동기

job·event

8

응답

status·trace

작업 증거 · 사용자에게 보인 결과

·지연에서도 책임과 다음 행동이 분명하다

15. 인쇄용 백엔드 처리 워크시트

15.1. 처리 범위

기능·업무 목표: _____
시작 HTTP 요청: _____
동기 응답이 보장하는 완료: _____
응답 뒤 남은 후속 작업: _____
최종 완료 증거: _____

15.2. 여덟 문

문	입력	판정·행동	통과 증거	실패·응답
접수				
검증				
신원·권한				
업무 규칙				
저장				
외부 연동				
비동기				
응답·관측				

15.3. 데이터와 후속 작업

transaction 안의 변경: _____
rollback 뒤 기대 상태: _____
외부 서비스·timeout: _____
재시도 가능한 조건·최대 횟수: _____
idempotency·업무 중복 키: _____
job·event 상태와 완료 통지: _____

15.4. 오류와 추적

```
problem type: _____________________________________________________
사용자가 고칠 detail·field: _____________________________________
instance·문의용 참조: ___________________________________________
trace ID·업무 record ID: ________________________________________
민감 정보 제외 규칙: ____________________________________________
```

16. 셀프 테스트 · 정답을 가리고 풉니다

문제 1

브라우저에서 제출 버튼을 비활성화했으므로 서버에서 권한 검사를 생략해도 될까요? 이유를 한 문장으로 씁니다.

문제 2

`quantity`가 문자열이라 실패한 경우와 정수 5지만 재고가 4라 실패한 경우는 각각 어느 검증 층입니까?

문제 3

로그인한 사용자가 다른 사용자의 주문 ID를 바꾸어 보냈습니다. 인증·인가·업무 규칙 중 어느 판정입니까?

문제 4

주문 취소 권한은 있지만 이미 배송이 시작됐습니다. 어느 판정입니까?

문제 5

재고 감소 뒤 주문 INSERT가 실패했습니다. rollback 뒤 재고와 주문 수는 어떻게 되어야 합니까?

문제 6

외부 결제 요청이 timeout 됐습니다. 곧바로 같은 요청을 무제한 반복하면 안 되는 두 이유를 씁니다.

문제 7

`202 Accepted`가 보장하는 것과 보장하지 않는 것을 각각 씁니다.

문제 8

비동기 작업 응답에 최소 세 가지 무엇을 넣어야 상태를 이어 볼 수 있습니까?

문제 9

RFC 9457 problem details에서 오류 종류와 이번 발생 건을 각각 식별하는 멤버는 무엇입니까?

문제 10

`detail` 문자열을 화면 로직이 파싱해서 분기하면 왜 위험합니까?

문제 11

trace·metric·log가 각각 답하는 대표 질문을 하나씩 씁니다.

문제 12

중복 클릭을 프런트 버튼 잠금과 백엔드 idempotency 양쪽에서 다루는 목적 차이를 씁니다.

17. 셀프 테스트 정답과 해설

정답 1

안 됩니다. 공격자는 브라우저 UI를 거치지 않고 직접 요청을 만들 수 있으므로 서버가 모든 객체·행동 권한을 최종 판정해야 합니다.

정답 2

문자열 `quantity`는 구조·타입 검증, 정수 5와 재고 4의 충돌은 서버 상태와 비교하는 불변 조건·업무 규칙입니다.

정답 3

인가, 더 구체적으로 객체 수준 권한 판정입니다. 신원은 이미 확인됐지만 해당 주문을 다룰 권한이 없습니다.

정답 4

업무 규칙입니다. 권한은 있어도 현재 주문 상태에서 취소가 허용되지 않습니다.

정답 5

트랜잭션 전 값으로 돌아가 재고와 주문 수가 모두 변하지 않아야 합니다. 부분 변경이 남으면 일관성이 깨집니다.

정답 6

원래 결제가 성공했지만 응답만 잃었을 수 있어 중복 결제가 생길 수 있고, 반복 요청이 외부 서비스 부하를 키울 수 있습니다. 결과 조회·idempotency·backoff·시도 한도가 필요합니다.

정답 7

처리 요청을 접수했다는 사실은 보장하지만 최종 작업 성공은 보장하지 않습니다. 상태 확인·완료 통지·실패 처리가 별도로 필요합니다.

정답 8

예: job ID, 현재 state, status URL입니다. 제출 시각·예상 확인 시점·취소 링크도 계약에 따라 더할 수 있습니다.

정답 9

오류 종류는 `type`, 이번 발생 건은 `instance`입니다.

정답 10

사람용 문장은 번역·문구 개선·발생 상황에 따라 바뀝니다. 기계 분기는 안정적인 `type`과 구조화 확장 필드를 써야 합니다.

정답 11

trace는 어디를 지나 어디서 느렸는지, metric은 얼마나 자주·얼마나 심한지, log는 이번 발생에 무슨 일이 있었는지를 답합니다.

정답 12

프런트 잠금은 사용자 피드백과 우발적 반복을 줄입니다. 백엔드 idempotency는 직접 호출·네트워크 재시도·여러 탭에서도 상태 변경 효과가 중복되지 않게 합니다.

18. 흔한 설명을 고쳐 씁니다

흔한 문장	문제	고쳐 쓴 문장
백엔드에서 처리합니다	단계·판정 없음	서버가 권한·재고 규칙 뒤 주문 tx를 commit합니다
validation 합니다	어느 층인지 없음	schema 통과 뒤 재고 불변 조건을 판정합니다
권한이 없습니다	신원·객체·업무 혼합	로그인됐지만 이 주문의 owner가 아니어서 403입니다
저장 실패입니다	부분 변경 여부 없음	INSERT 오류로 tx가 rollback돼 before 값입니다
비동기로 돌립니다	완료·실패 계약 없음	202와 job ID를 주고 worker 완료를 status URL에서 확인합니다
retry 합니다	중복·한도 없음	timeout만 최대 2회 backoff하며 key로 중복 효과를 막습니다
로그를 확인합니다	찾는 방법 없음	trace ID로 API span·payment attempt·rollback log를 연결합니다

19. 최종 제출 체크리스트

- [ ] 클라이언트와 서버 사이 신뢰 경계를 표시했습니다.

- [ ] 요청 method·path·schema·크기·추적 ID를 적었습니다.
- [ ] 문법·구조·의미·불변 조건 검증을 구분했습니다.
- [ ] 인증·객체 인가·업무 규칙의 판정 근거를 분리했습니다.
- [ ] transaction에 포함할 변경과 rollback 뒤 값을 적었습니다.
- [ ] 외부 호출의 timeout·retry·중복·fallback을 적었습니다.
- [ ] 202가 있다면 job ID·상태·완료 확인 방법을 적었습니다.
- [ ] status·problem type·detail·instance를 계약했습니다.
- [ ] trace ID와 업무 record ID로 증거를 연결했습니다.
- [ ] 성공·검증·권한·충돌·저장·외부·비동기·중복 시나리오를 검수했습니다.
- [ ] 개인정보·secret·token을 오류 응답과 로그에서 제외했습니다.
- [ ] 프런트 표시 책임과 백엔드 최종 판정 책임을 구분했습니다.

20. 공식 근거와 다음 학습

이 교재는 다음 공식 자료를 2026-07-15에 확인해 학습자용으로 재구성했습니다.

- RFC 9110 · HTTP Semantics: request·response, method, status, safe·idempotent, 202 의미
- RFC 9457 · Problem Details for HTTP APIs: 오류 객체와 멤버·보안 고려
- OWASP API Security Top 10 2023 · Object Level Authorization: 객체 ID를 받는 기능의 권한 검사
- OWASP Secure Coding Practices · Input validation: 신뢰 시스템의 서버 검증
- OWASP ASVS: 웹 애플리케이션 보안 검증 요구사항
- PostgreSQL · Transactions: all-or-nothing, commit·rollback·가시성
- OpenAPI Specification: HTTP API 계약 기술
- CloudEvents Specification: event 문맥과 데이터의 공통 형식
- OpenTelemetry · Signals: traces·metrics·logs
- OpenTelemetry · Context propagation: 서비스 경계의 신호 연관

다음 M04-02에서는 API endpoint를 리소스·행동·요청·성공·오류 계약으로 더 구체화합니다. M04-03에서는 인증·인가·세션·token을, M04-04에서는 schema와 transaction을, M04-05에서는 로그와 예외 추적을 깊게 다룹니다.

M04-01 완료
이제 “서버가 알아서 처리한다”는 검은 상자를 여덟 문으로 열 수 있습니다. 요청 하나를 골라 각 문에서 무엇을 믿지 않고, 누가 판정하고, 무엇이 바뀌며, 실패하면 어떤 상태와 증거가 남는지 말하면 다음 API 설계로 갈 준비가 됐습니다.