M08-03 · GIBALJA VISUAL MANUAL

실제 서비스처럼 오류와 상태 처리하기

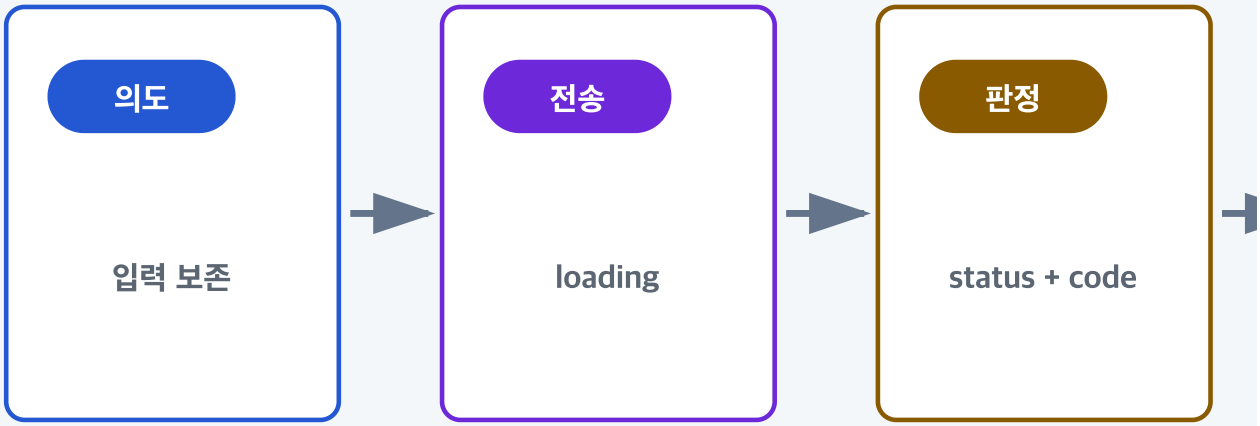
> **한 문장 목표:** `요청 의도 → loading → 통신·HTTP 판정 → 화면 상태 → 입력·기존 데이터 보존 → 상황별 복구 → trace·fault evidence`를 한 번 연결하고, 재시도가 중복 부수 효과를 만들지 않음을 증명합니다.

난이도	개념	실습	셀프 테스트	최종 산출물
Level 2	70분	70분	15분	복구 가능한 화면, 오류 계약, fault matrix, idempotency 증거 packet

오류 처리는 빨간 문장을 하나 띄우는 일이 아닙니다. 사용자가 **지금 무엇이 일어났고, 입력은 남아 있으며, 다음에 무엇을 해야 하는지** 알게 만들고, 시스템은 **같은 실패와 복구를 다시 증명**할 수 있어야 합니다. 이번 실습의 계정·데이터·오류는 모두 합성입니다.

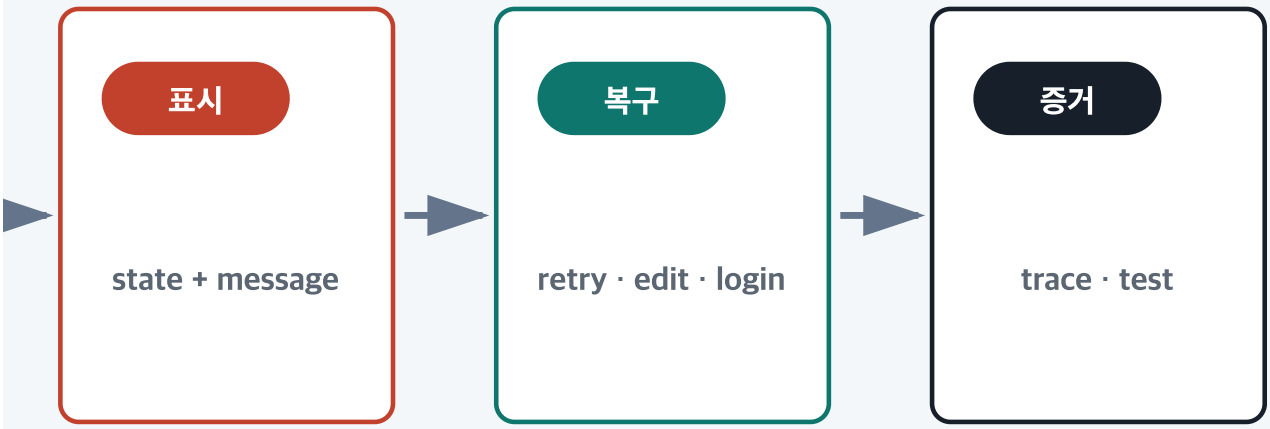
그림 1. 오류 처리는 `catch` 한 줄이 아니라 요청을 보내기 전의 입력 보존부터 복구·증거까지 이어지는 생명주기입니다.

보내기 전부터 복구와 증거까지 한 줄로 연결한다.



완료

사용자는 지금 상태와 다음 행동을 알고



기준

, 시스템은 같은 결과를 재현할 수 있다.

0. 이 PDF를 공부하는 방법

1회차 · 그림만 읽기 · 25분

그림 1부터 그림 16까지 제목과 하단 결론만 읽습니다. 다음 여덟 문장을 말할 수 있으면 첫 회차는 완료입니다.

```
통신 실패는 HTTP status가 없는 실패다.
HTTP 실패는 Response가 왔지만 response.ok가 false인 실패다.
Problem Details의 code는 화면 분기, detail은 사람의 수정 행동을 돕는다.
loading, empty, ready, error는 서로 다른 화면 약속이다.
422 뒤에는 입력을 지우지 않고 수정할 필드로 안내한다.
409는 최신 상태를 먼저 읽고, 429는 Retry-After를 기다린다.
POST를 다시 보낼 때는 같은 idempotency key로 한 번의 결과를 확인한다.
취소는 서버 부수 효과를 되돌린다는 뜻이 아니며, 오래된 응답은 무시한다.
```

2회차 · 실제 오류 화면 비교 · 20분

실습 생성기를 실행합니다.

```
./02_Labs/G08_Fullstack/L08-03_create-resilient-error-state-practice.sh
```

생성기는 외부 package와 network 없이 다음 묶음을 만듭니다.

```
gibalja-resilient-error-state-practice/
├── app/       로그인·권한·오류 주입·복구가 있는 local 앱
└── evidence/  reset·40 tests·25 audit·fault probe·state contract·version
```

3회차 · 장면을 하나씩 재현 · 55분

오류 실습 패널에서 다음 순서로 실행합니다.

```
정상 → 빈 목록 → 느린 응답 취소 → 통신 끊김
→ 422 입력 오류 → 409 변경 충돌 → 429 요청 제한
→ 503 저장 장애 → 저장 후 응답 시간 초과 → 응답 경쟁
```

매 장면에서 다섯 가지만 기록합니다.

확인	질문
HTTP	응답이 있었는가, status는 무엇인가
code	화면이 읽을 안정적인 분기 값은 무엇인가
state	화면은 loading·empty·ready·error 중 무엇인가
preservation	입력과 기존 목록이 남았는가
recovery	수정·로그인·최신본·대기·재시도 중 무엇인가

4회차 · 내 기능에 적용 · 55분

오류·상태·복구 기록 양식을 채우고 단계별 실습서를 따릅니다. 낯선 표현은 오류·상태·복구 용어집에서 찾습니다.

1. 이번 Vertical Slice의 결과와 비목표를 고정합니다

M08-02에서 관찰 보드는 로그인·세션·권한을 얻었습니다. M08-03에서는 같은 기능을 성공 경로만 있는 데모에서 실패를 견디고 복구할 수 있는 서비스 표면으로 바꿉니다.

1.1. 검증할 outcome

```
actor: 합성 planner
starting_state:
  - login session active
  - synthetic observations: 2
  - draft form: empty
actions:
  - load fault scenario
  - submit fault scenario
visible_result:
  - explicit UI state
  - safe message and next action
  - trace or local reference
preservation:
  - existing list remains on refresh failure
  - draft remains on submit failure
side_effect:
  - retry with same idempotency key creates one row only
```

1.2. 의도적 non-goal

제외	이유	운영으로 갈 때
실제 장애·실제 고객 data	학습 안전 경계	staging fault injection과 data governance
multi-region retry	분산 합의·복제 범위	region별 idempotency·consistency 설계
message queue·background job	동기 HTTP slice에 집중	job state·dead letter·reconciliation
중앙 trace backend	package·network 없는 실습	OpenTelemetry·vendor trace와 sampling
production rate limiter	분산 counter·정책 제외	actor·resource별 quota와 abuse control
service worker offline queue	충돌·동기화 범위 확대	offline-first product contract
자동 병합 UI	409 읽기·해결에 집중	version·diff·merge policy
무한 자동 retry	장애 증폭 위험	bounded backoff·jitter·retry budget

로컬 전용 장치
`X-Lab-Fault` 는 합성 오류를 재현하는 교육용 header입니다. production API에 이 기능을 두지 않습니다. 실습 server는 `127.0.0.1` 에서만 실행하고 실제 계정·개인정보·운영 URL을 넣지 않습니다.

2. 요청 생명주기를 화면·API·DB로 펼칩니다

오류를 `try/catch` 위치만으로 보면 화면과 DB의 상태를 놓칩니다. 요청 전·중·후를 한 줄로 기록합니다.

단계	browser	API	DB·외부 효과	남길 증거
의도	입력·기존 목록 보존	아직 요청 없음	변화 없음	draft snapshot
전송	loading·button disabled	request 수신	아직 모름	method·path·request id
판정	network 또는 response	status·problem 생성	commit 여부	status·code·trace
표시	ready·empty·error	응답 완료	결과 존재 가능	screenshot·state
복구	edit·login·reload·wait·retry	새 request	중복 방지 필요	same key·row count
증거	최종 화면	contract probe	DB snapshot	evidence packet

2.1. 실패해도 먼저 지킬 것

```
submit 시작 전 draft를 읽을 수 있어야 한다.
refresh 실패 중 기존 목록을 즉시 지우지 않는다.
성공이 확인되기 전 form.reset()을 호출하지 않는다.
서버 commit 여부가 모르면 새 저장으로 단정하지 않는다.
```

3. 통신 실패와 HTTP 실패를 먼저 구분합니다

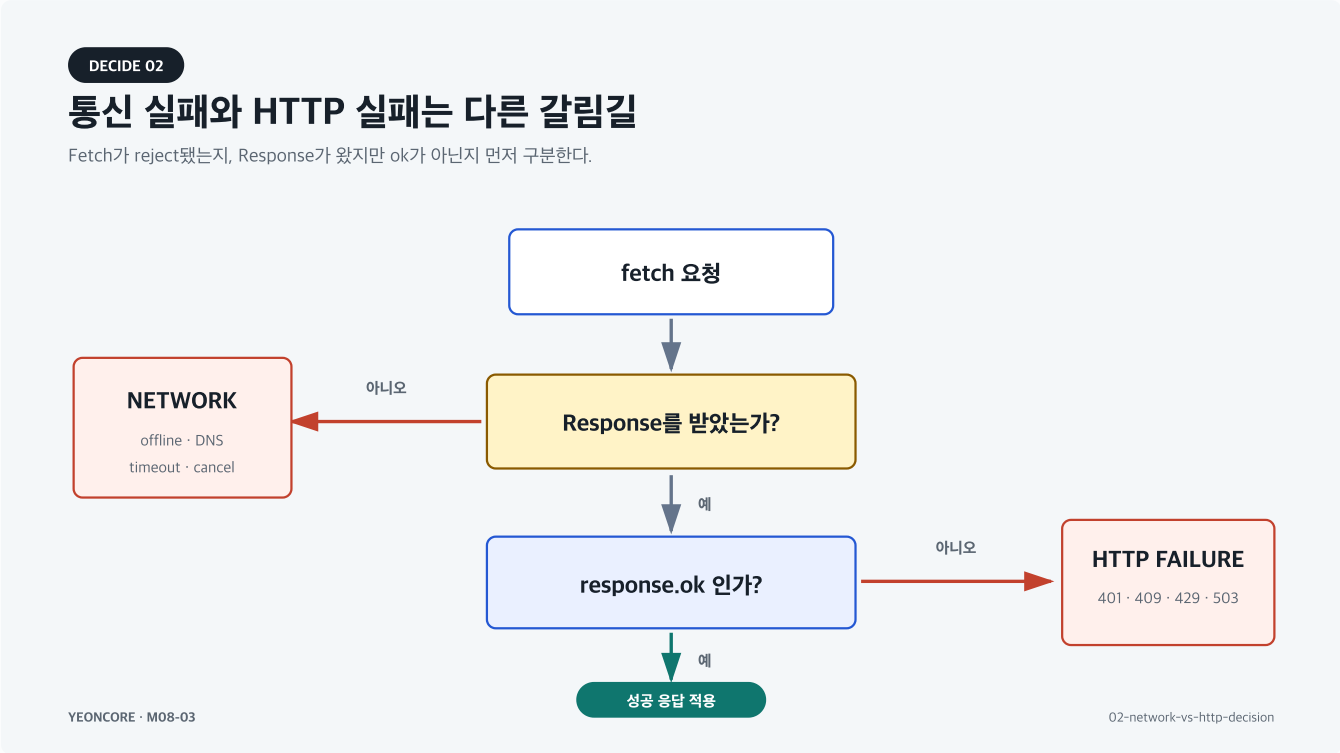
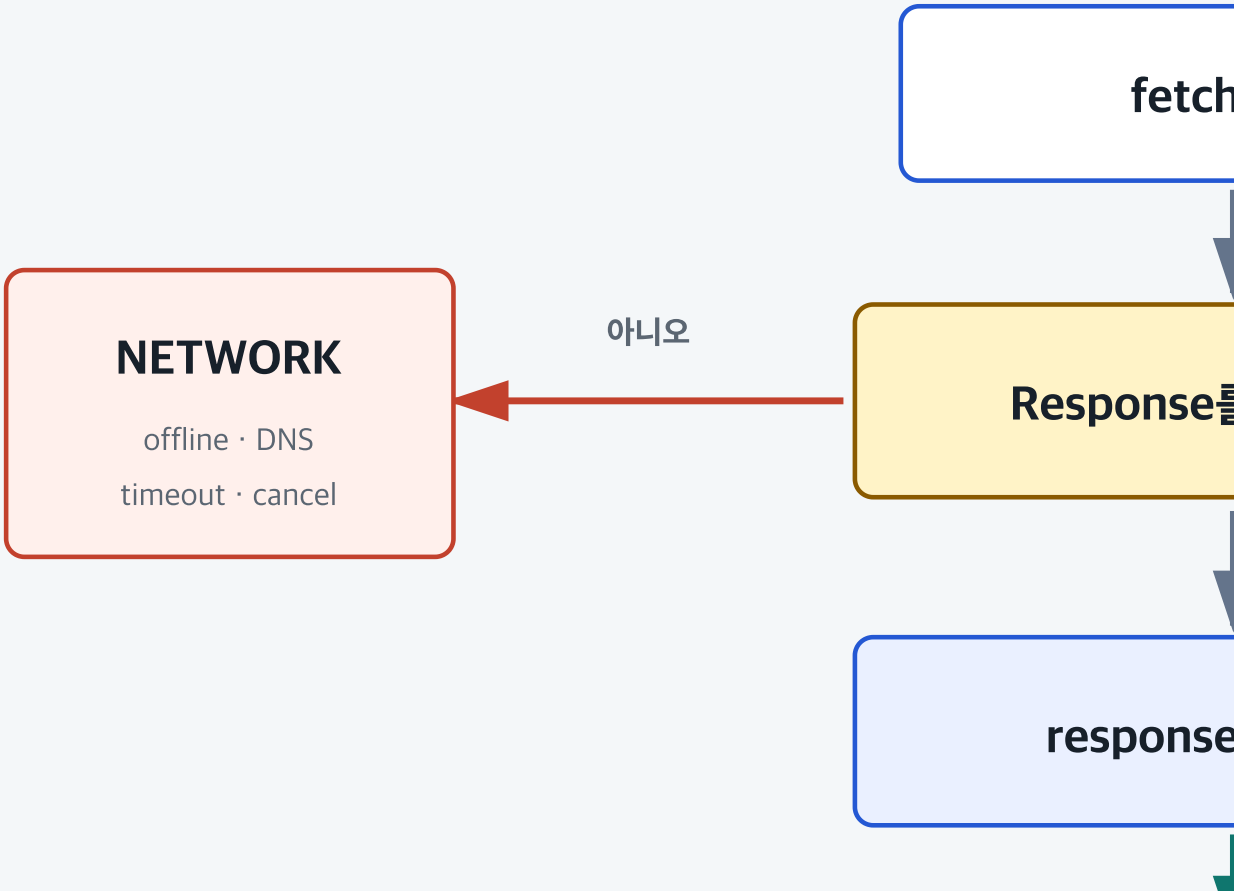
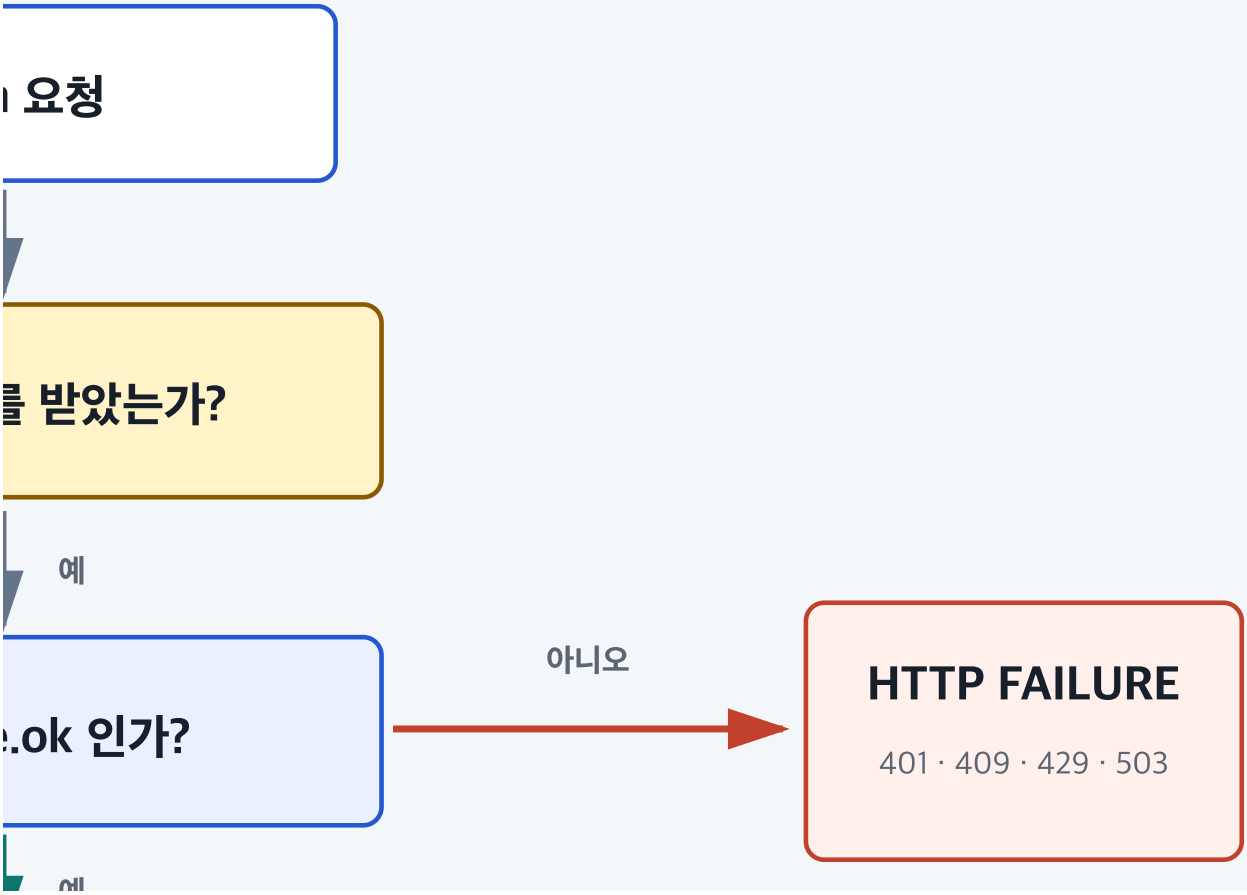


그림 2. 통신 실패에는 HTTP status가 없습니다. HTTP 401·409·429·503은 응답을 받은 뒤 `response.ok`로 판정합니다.

Fetch가 reject됐는지, Response가 왔지만 ok가 아닌지 먼저 구분한다.




요청
받았는가?
예
.ok 인가?
아니오
HTTP FAILURE
401 · 409 · 429 · 503

3.1. Fetch의 두 실패 경로

```
let response;
try {
  response = await fetch(url, options);
} catch (error) {
  // network, DNS, CORS network error, abort, browser-level failure
}

if (!response.ok) {
  // HTTP response arrived, but status is outside 200..299
}
```

`fetch()`가 resolve되었다고 업무 성공은 아닙니다. `404`, `422`, `503`도 `Response` 객체로 돌아옵니다.

3.2. 네 종류를 같은 문장으로 뭉치지 않습니다

종류	Response	status	예	기본 다음 행동
network failure	없음	없음	DNS·연결 끊김	연결 확인·수동 retry
abort	없음	없음	사용자 취소·timeout signal	취소 이유 표시·결과 불명 분리
HTTP failure	있음	4xx·5xx	409·422·429·503	status·code별 행동
protocol failure	있음	다양	JSON 계약 파손	안전한 fallback·trace

30초 확인 1

서버가 503 Problem Details를 반환했습니다. 이 요청은 Fetch의 `catch`로만 처리해야 합니까, 아니면 `response.ok`와 status를 읽어야 합니까?

4. Problem Details로 오류 계약을 고정합니다

그림 3. 표준 필드는 오류의 공통 골격을 만들고, `code`·`errors`·`retryable` 같은 extension은 화면 행동을 안정적으로 연결합니다.

사람이 읽는 설명과 기계가 읽는 code를 섞지 않는다.

```
{
"type": ".../validation-failed",
"title": "입력값을 확인해 주세요",
"status": 422,
"detail": "수정할 항목이 있습니다",
"instance": "urn:trace:req_83f1",
"code": "VALIDATION_FAILED",
"trace_id": "req_83f1",
"retryable": false,
"errors": [{"field":"summary", ...}]
}
```

HTTP 의미

status

화면 분기

code · errors

사용자 복구

title · detail

지원·로그 연결

instance · trace_id

4.1. 실습 오류 응답

```
HTTP/1.1 422 Unprocessable Content
Content-Type: application/problem+json; charset=utf-8
X-Trace-ID: req_c35495fce5d38369
Cache-Control: no-store

{
  "type": "https://problems.example.invalid/validation-failed",
  "title": "입력값을 확인해 주세요",
  "status": 422,
  "detail": "표시된 항목을 수정하세요.",
  "instance": "urn:trace:req_c35495fce5d38369",
  "code": "VALIDATION_FAILED",
  "trace_id": "req_c35495fce5d38369",
  "retryable": false,
  "errors": [
    {
      "field": "summary",
      "code": "INVALID_FIELD",
      "message": "관찰 근거를 더 구체적으로 적어 주세요."
    }
  ]
}
```

`example.invalid`는 실습용으로 실제 연결되지 않는 domain입니다. production은 통제하는 domain 아래에 problem type 문서를 게시한 뒤 안정적인 URI를 사용합니다.

4.2. 필드별 책임

field	독자	변해야 하는가	사용
`type`	사람·기계	problem 종류마다 안정	의미 문서 URI
`title`	사람	occurrence마다 바꾸지 않음	짧은 종류 이름
`status`	HTTP 소비자	실제 status와 일치	일반 분기
`detail`	사람	occurrence마다 가능	해결을 돕는 설명
`instance`	지원·forensic	발생마다 고유	한 번의 오류 식별
`code`	app UI	안정	화면 state·action 분기
`errors`	form UI	field마다 다름	inline message 연결

field	독자	변해야 하는가	사용
`retryable`	retry UI	조건별	버튼 제공 여부

4.3. `detail`을 parsing하지 않습니다

```
// 나쁨: 번역·문장 변경에 깨짐
if (problem.detail.includes("3초")) retry();

// 좋음: 안정적인 값 사용
if (problem.code === "RATE_LIMITED" && problem.retryable) {
  showRetry(problem.retry_after_seconds);
}
```

5. 오류 Taxonomy와 책임자를 연결합니다

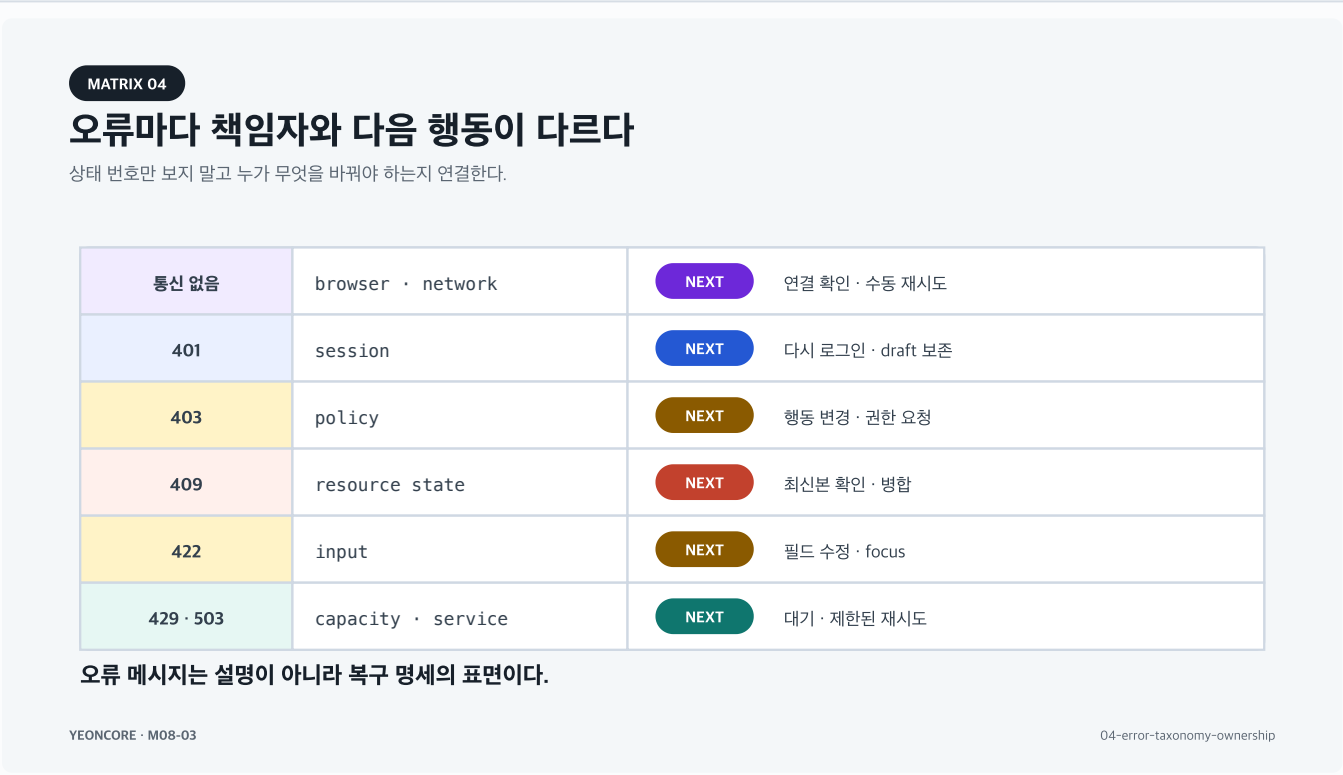


그림 4. status는 원인 위치를 좁히고, owner와 next action을 붙여야 실제 복구 명세가 됩니다.

상태 번호만 보지 말고 누가 무엇을 바꿔야 하는지 연결한다.

통신 없음	browser · network
401	session
403	policy
409	resource state
422	input
429 · 503	capacity · service

오류 메시지는 설명이 아니라 복구 명세의 표면이다

NEXT	연결 확인 · 수동 재시도
NEXT	다시 로그인 · draft 보존
NEXT	행동 변경 · 권한 요청
NEXT	최신본 확인 · 병합
NEXT	필드 수정 · focus
NEXT	대기 · 제한된 재시도

5.1. 이 실습의 오류·상태·복구 행렬

장면	HTTP	code	UI state	입력	기존 목록	다음 행동
통신 끊김	없음	`NETWORK_ERROR`	error	유지	유지	연결 확인·retry
session 종료	401	`AUTHENTICATION_REQUIRED`	signed out	유지	숨김	login
permission 부족	403	`PERMISSION_DENIED`	error/read-only	유지	유지	행동·역할 변경
변경 충돌	409	`VERSION_CONFLICT`	conflict	유지	유지	최신 목록 확인
입력 오류	422	`VALIDATION_FAILED`	field_error	유지	유지	field 수정
요청 제한	429	`RATE_LIMITED`	rate_limited	유지	유지	Retry-After 대기
일시 장애	503	`SERVICE_UNAVAILABLE`	error	유지	유지	제한된 retry
응답 파손	502	`INVALID_RESPONSE`	error	유지	유지	fallback·지원

5.2. 5xx를 전부 자동 재시도하지 않습니다

5xx는 server-side 실패 범주이지만 요청이 적용되었는지는 별도 질문입니다. 특히 POST는 응답을 못 받았더라도 DB commit이 끝났을 수 있습니다.

6. 목록과 저장을 두 개의 상태 기계로 나눕니다

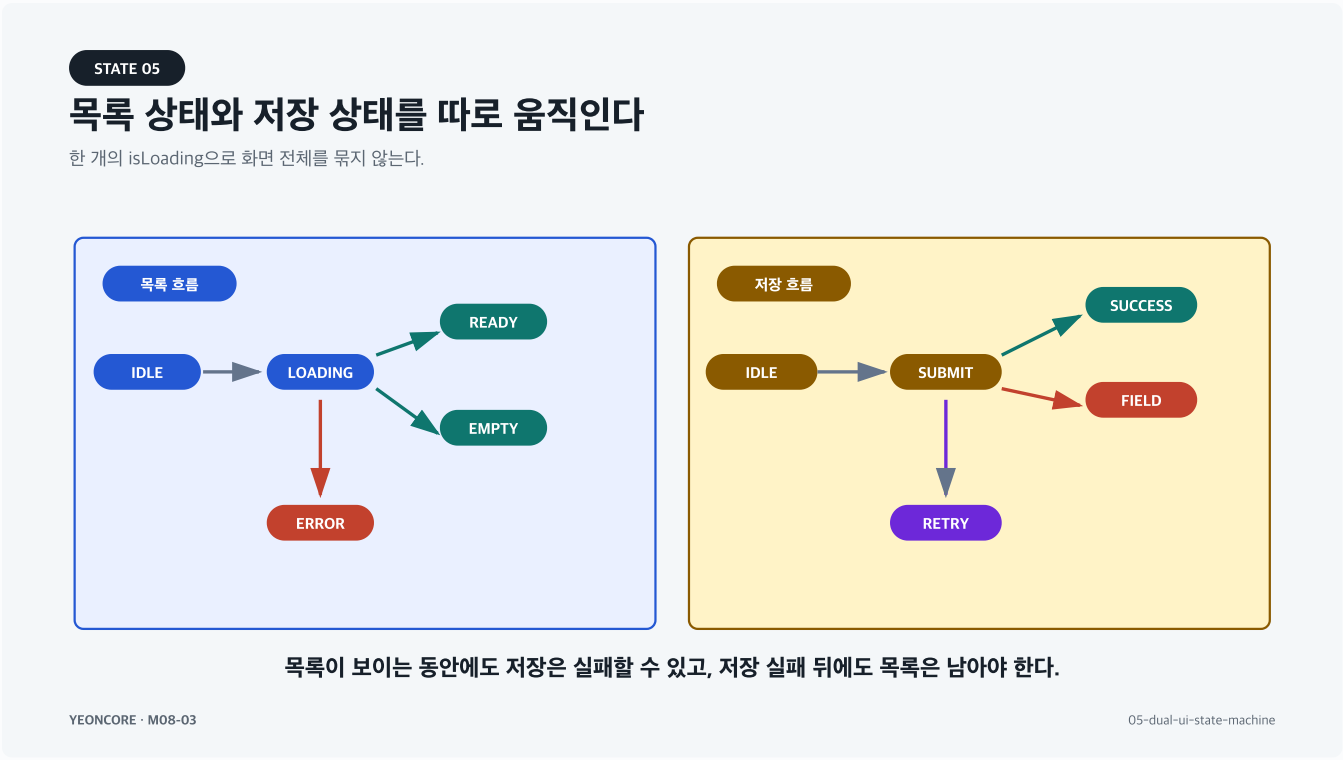
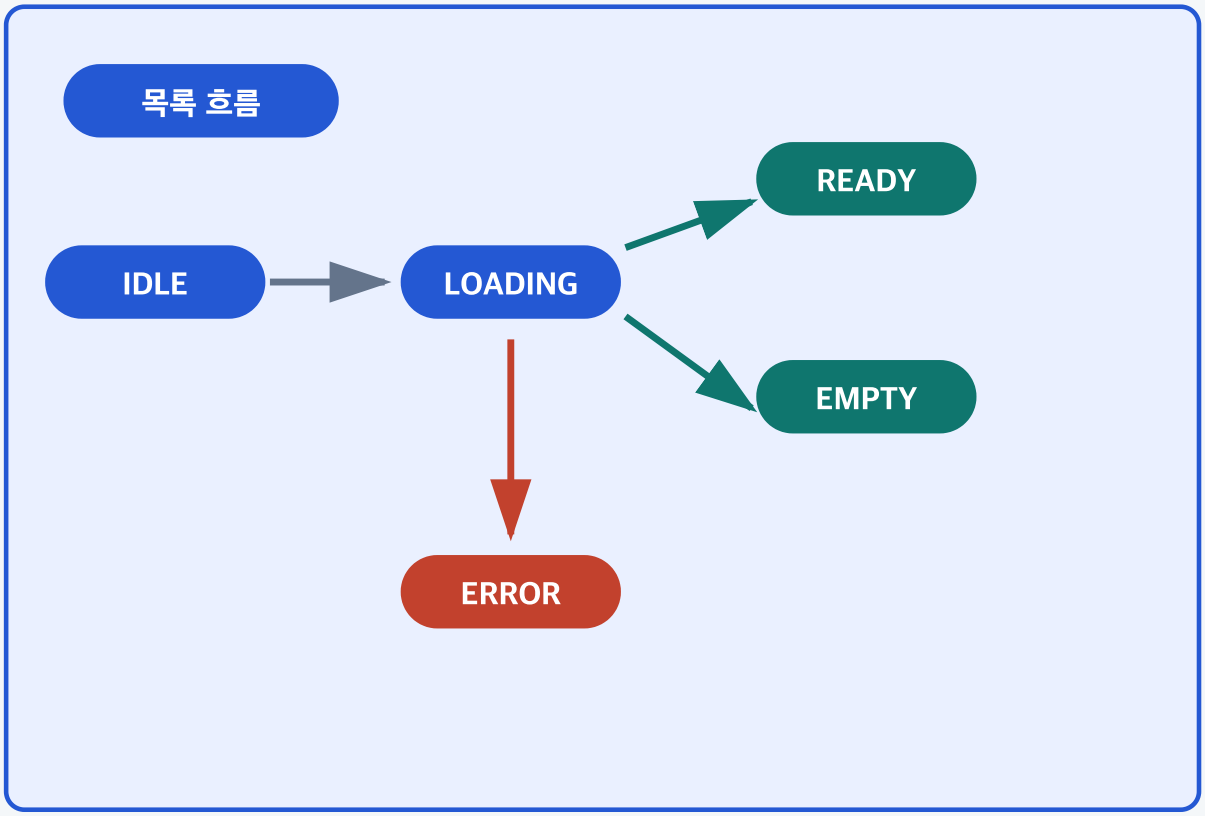
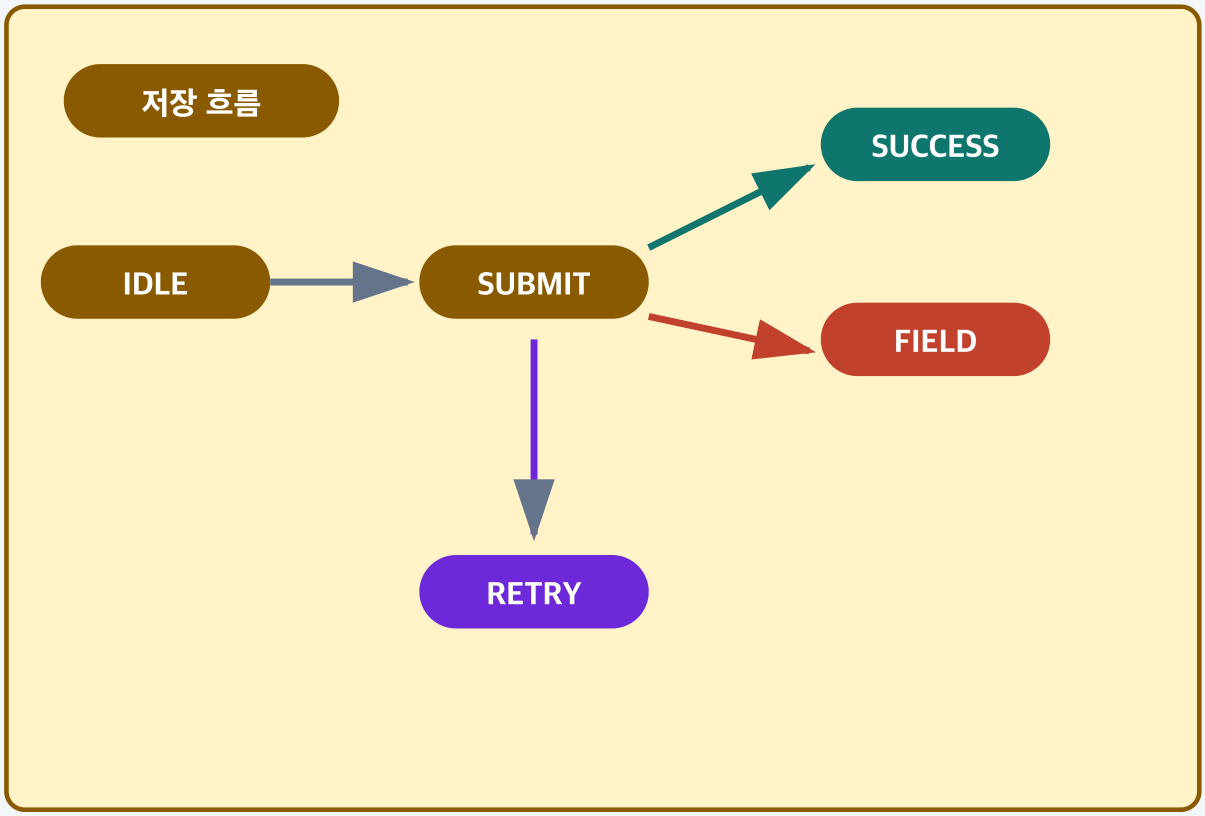


그림 5. 목록이 ready인 동안 저장은 submitting·field_error·conflict일 수 있습니다. 하나의 `isLoading`으로 표현할 수 없습니다.

한 개의 isLoading으로 화면 전체를 묶지 않는다.




저장 흐름
IDLE
SUBMIT
SUCCESS
FIELD
RETRY

6.1. 목록 state

```
idle → loading → ready
               ├→ empty
               ├→ error
               └→ canceled
```

6.2. 저장 state

```
idle → submitting → success
                  ├→ field_error
                  ├→ conflict
                  ├→ rate_limited
                  └→ error
```

6.3. 상태와 데이터는 함께 기록합니다

```
const state = {
  observations: [],
  listState: "idle",
  pendingSubmission: null,
  latestListRequest: 0,
};
```

`error` 하나만 저장하면 어떤 입력을 다시 보내야 하는지, 이전 목록을 유지할지 알 수 없습니다.

7. loading·empty·ready를 서로 다른 화면으로 만듭니다

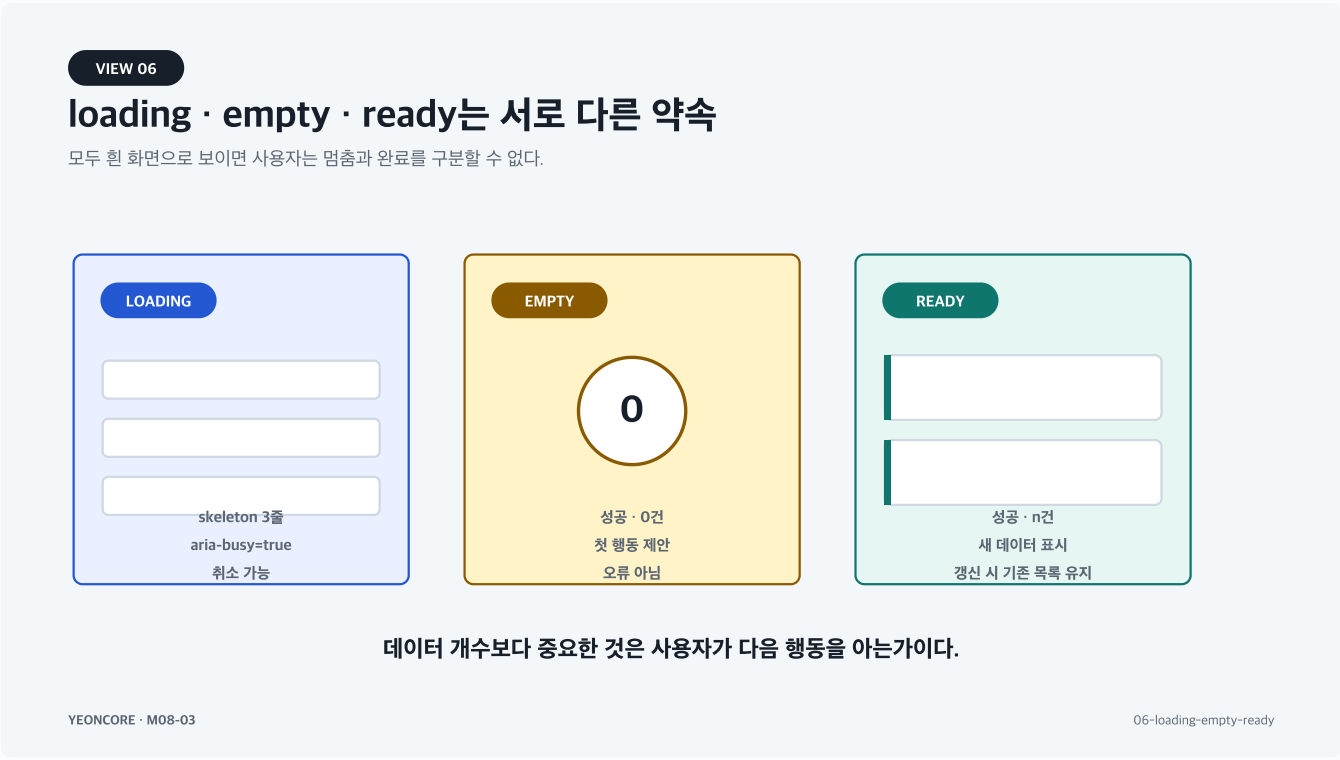
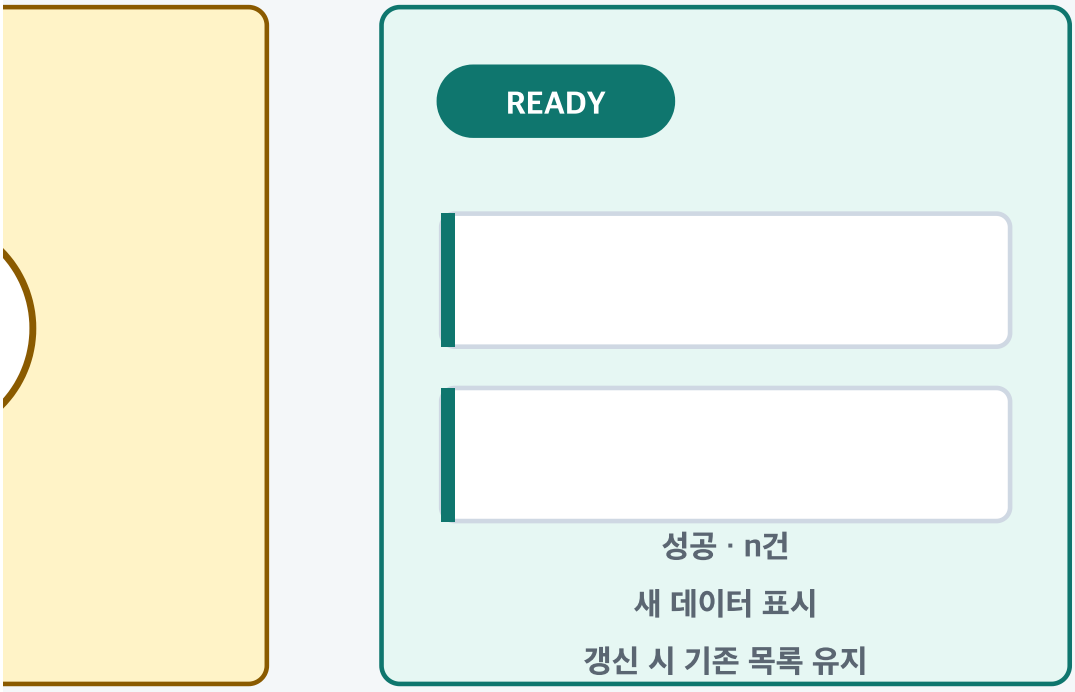


그림 6. `empty`는 성공한 0건이지 오류가 아닙니다. loading은 기다림과 취소 가능성을 보여 줍니다.

모두 흰 화면으로 보이면 사용자는 멈춤과 완료를 구분할 수 없다.

데이터 개수보다 중요한 것은 사



용자가 다음 행동을 아는가이다.

7.1. Rendering contract

state	보여 줄 것	숨기지 않을 것	접근성
loading	stable skeleton·진행 문장	기존 목록 또는 공간	`aria-busy=true`
empty	0건 설명·첫 행동	제목·filter context	status message
ready	최신 rows·count	refresh action	semantic list
error	안전한 설명·다음 행동	실패 전 데이터	alert·focus
canceled	취소 결과·다시 요청	기존 데이터	polite status

7.2. 빈 화면과 빈 목록은 다릅니다

```
나쁨: data.length === 0 → 아무것도 render하지 않음
좋음: data.length === 0 → "아직 관찰이 없습니다. 첫 관찰을 저장하세요."
```

8. 422는 입력을 보존하고 필드로 안내합니다

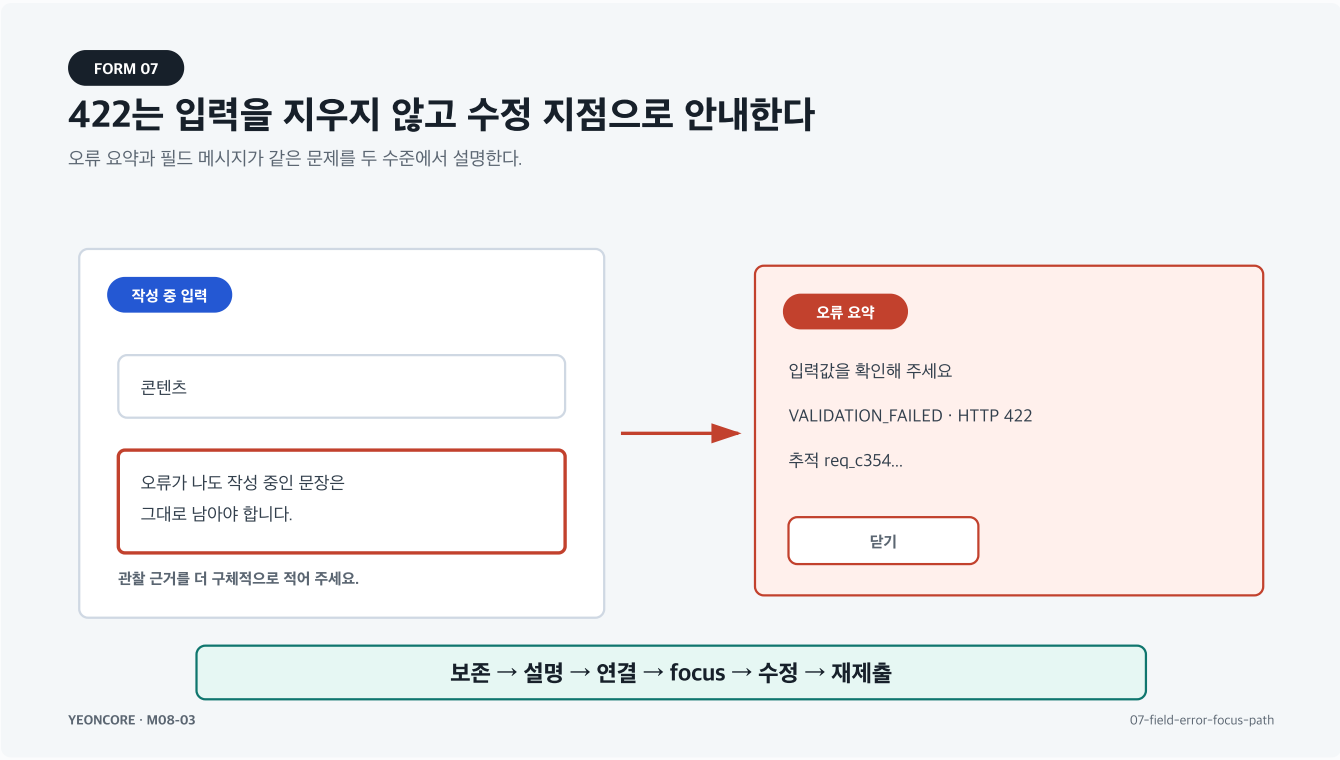
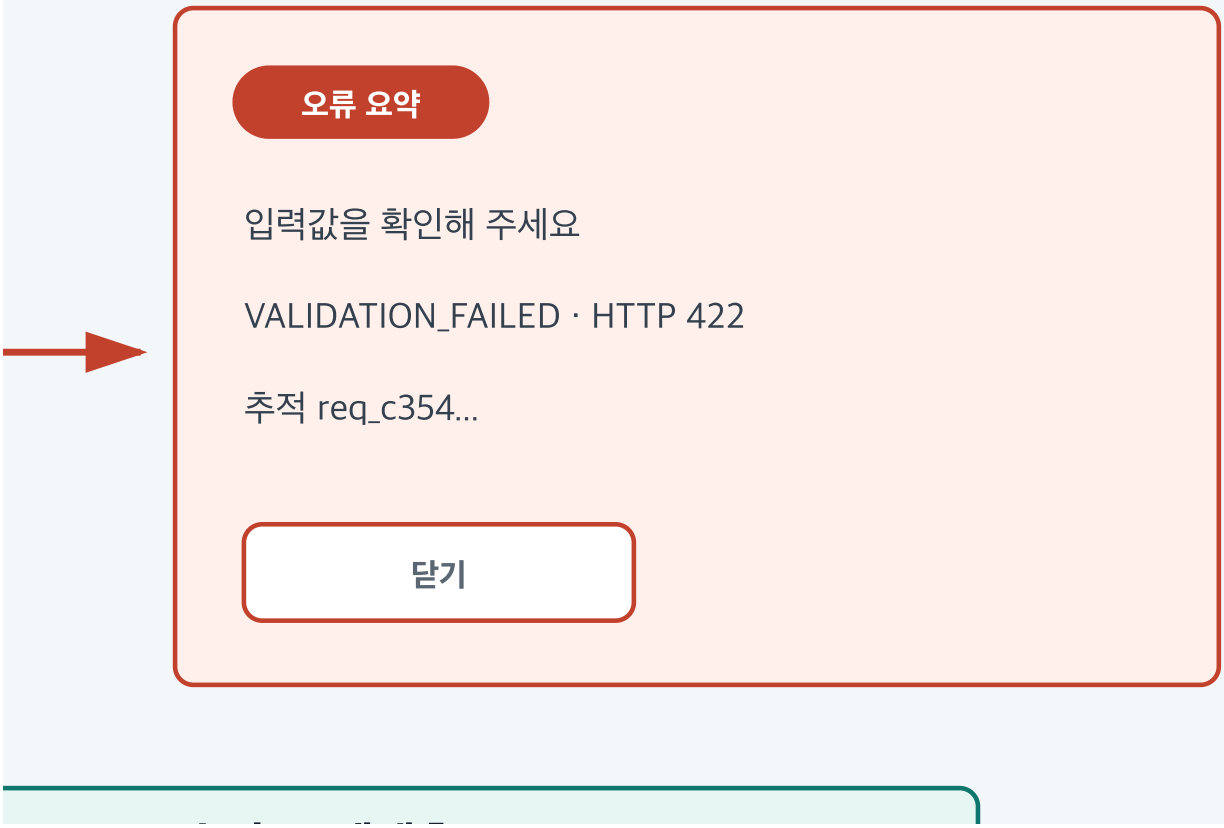


그림 7. 오류 요약은 전체 실패를 알리고, inline message는 고칠 위치를 알려 줍니다. 성공 전에는 입력을 지우지 않습니다.

오류 요약과 필드 메시지가 같은 문제를 두 수준에서 설명한다.


오류 요약
입력값을 확인해 주세요
VALIDATION_FAILED · HTTP 422
추적 req_c354…
닫기

8.1. 성공 경로에만 reset을 둡니다

```
try {
  const created = await requestJson("/api/observations", options);
  setState("success");
  form.reset();
} catch (error) {
  applyFieldErrors(error.errors);
  // reset 없음: 사용자가 쓴 값 유지
}
```

8.2. 필드와 오류 문장을 programmatically 연결합니다

```
<textarea
  id="summary"
  aria-describedby="summary-help summary-error">
</textarea>
<p id="summary-error" class="field-error"></p>
```

```
summaryInput.setAttribute("aria-invalid", "true");
summaryError.textContent = issue.message;
```

8.3. 실제 422 화면

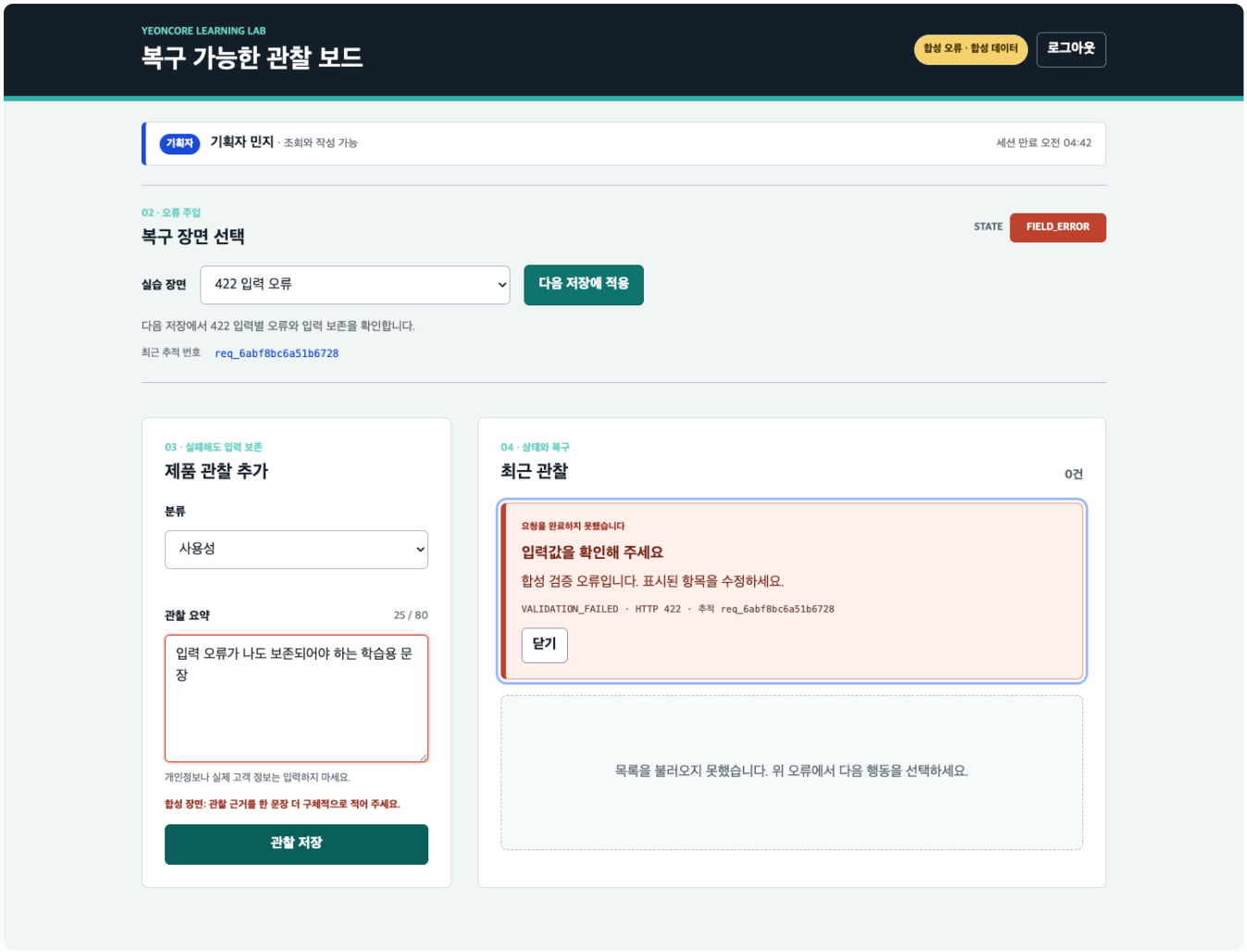


그림 8. 합성 422 뒤 작성 중 문장 31자가 그대로 남고, 기존 관찰 2건도 유지됩니다. 오류 요약·field border·inline message가 같은 문제를 설명합니다.

30초 확인 2

422 뒤 사용자가 다시 30자를 입력해야 한다면 무엇이 실패한 것인가요? Server validation입니까, 아니면 client preservation contract입니까?

9. 401과 403의 복구 행동을 나눕니다

그림 9. 401은 identity context를 복구하고, 403은 현재 identity가 할 수 있는 행동이나 policy를 바꿉니다.

둘 다 빨간 오류로 끝내면 사용자는 무엇을 바꿔야 할지 모른다.

401

세션이 없거나 종료됨

draft 보존

로그인 화면으로 이동

로그인 뒤 원래 맥락 복원

403

신원은 알지만 행동이 금지됨

현재 화면 유지

허용된 행동 제시

필요하면 권한 요청 경로

9.1. 401 · 다시 인증하되 draft를 잃지 않습니다

```
session cookie expired
→ protected request 401
→ form draft는 DOM·safe state에 남김
→ login view
→ 로그인 성공 뒤 원래 task로 복귀
```

9.2. 403 · 다시 로그인만 시키지 않습니다

viewer가 다시 viewer로 로그인하면 permission은 바뀌지 않습니다.

403 원인	안내
role에 action 없음	허용된 행동·권한 요청 경로
CSRF proof 실패	안전한 새 session·화면 reload
resource ownership 불일치	자신의 resource 또는 담당자
조직 policy 제한	승인 조건·owner

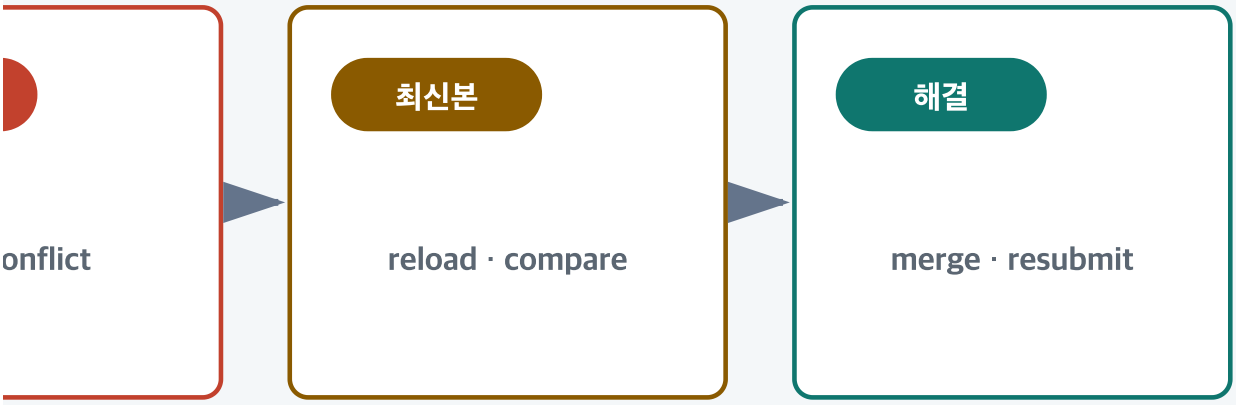
10. 409는 최신 상태를 읽고 해결합니다

그림 10. 409는 일시적 network 오류가 아니라 현재 resource state와 충돌한 요청입니다. 같은 요청을 즉시 반복하지 않습니다.

같은 입력을 다시 보내도 충돌 원인이 사라지지 않을 수 있다.

금지된

409을 받은 같은 요청



자동화

을 조건 없이 반복하기

10.1. 복구 flow

```
내가 본 version 1
→ 다른 사용자가 version 2 저장
→ 내 version 1 기반 저장
→ 409 VERSION_CONFLICT
→ 최신 version 2 불러오기
→ 차이 확인·내 입력 보존
→ merge 또는 포기
→ 새 조건으로 재제출
```

10.2. production conditional request

실제 update API는 ETag·`If-Match` 같은 conditional request를 검토할 수 있습니다. 실습은 create 기능에 합성 409를 주입해 UI 복구 흐름만 다룹니다.

11. 429·503은 Retry-After와 재시도 예산을 읽습니다

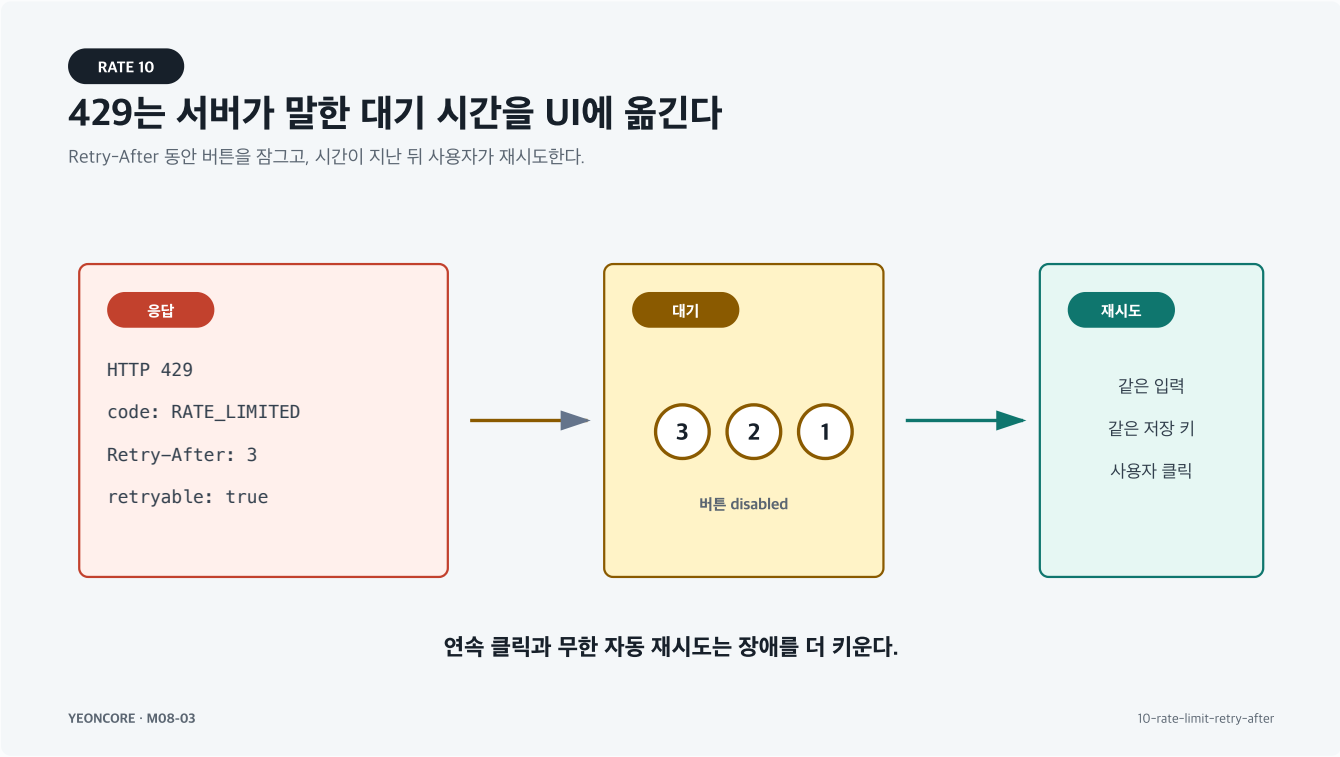
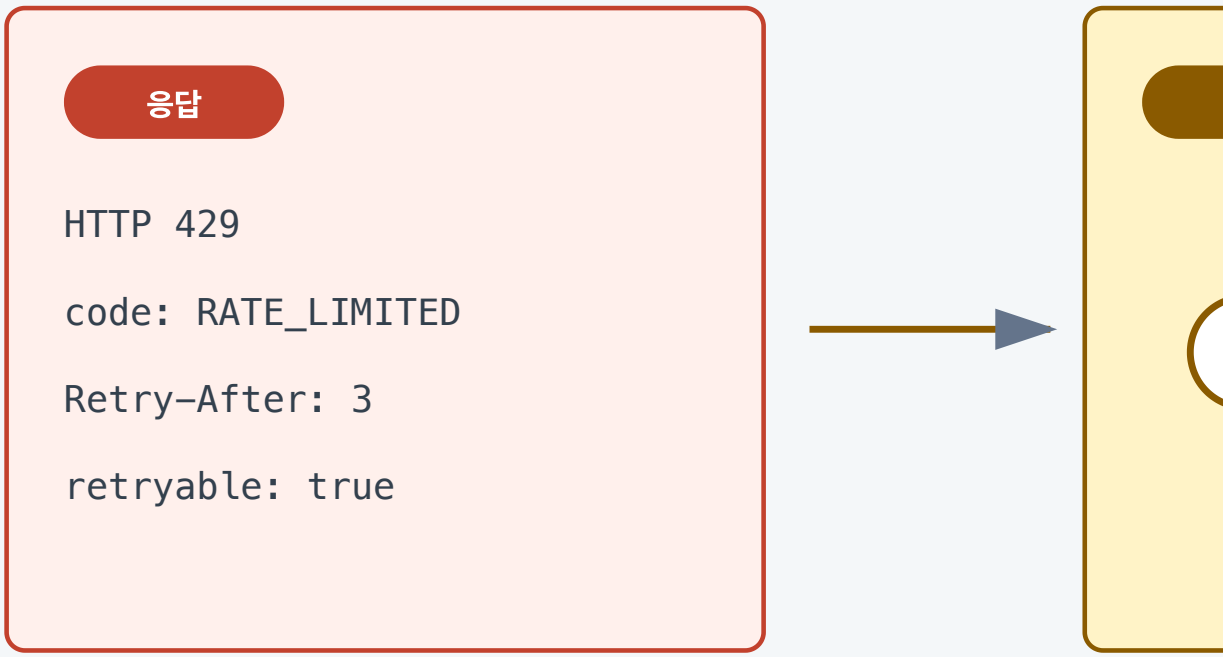
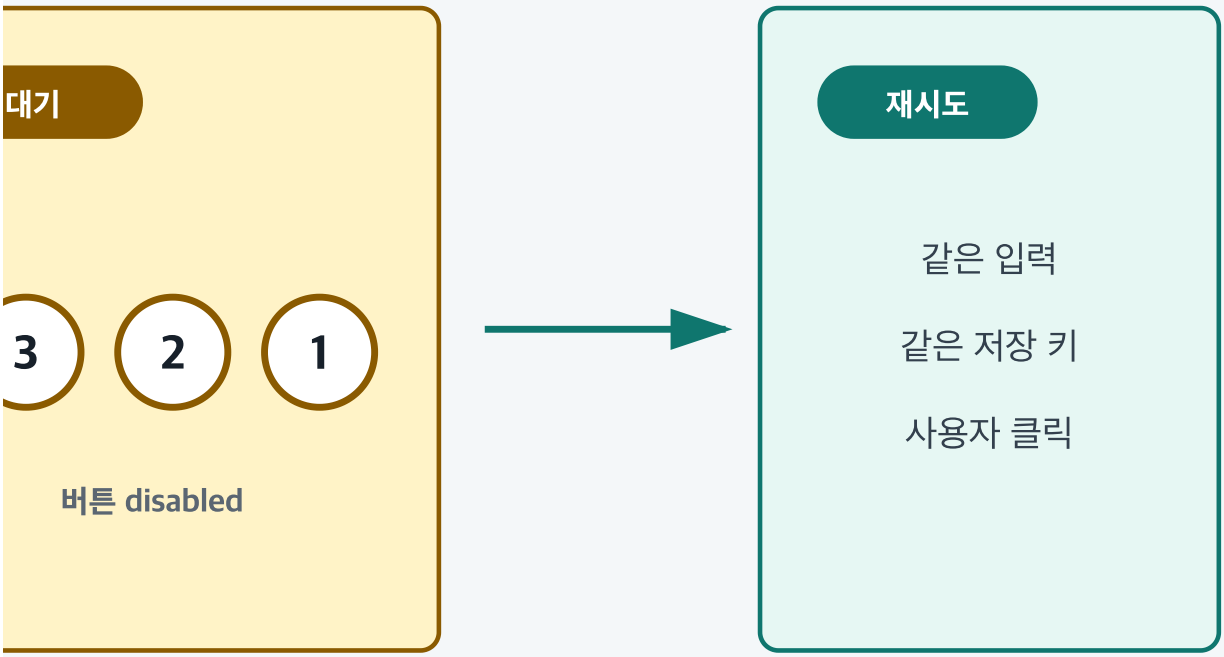


그림 11. 429는 사용자의 요청 빈도가 제한되었음을 뜻합니다. 서버의 대기 신호를 화면 countdown과 button state로 옮깁니다.

Retry-After 동안 버튼을 잠그고, 시간이 지난 뒤 사용자가 재시도한다.



연속 클릭과 무한 자동 재



시도는 장애를 더 키운다.

11.1. 429 실습 응답

```
HTTP/1.1 429 Too Many Requests
Retry-After: 3
Content-Type: application/problem+json

{
  "code": "RATE_LIMITED",
  "retryable": true,
  "retry_after_seconds": 3
}
```

11.2. Retry-After 읽기

```
const header = response.headers.get("Retry-After");
const seconds = Number.parseInt(header, 10);
```

HTTP `Retry-After`는 seconds 또는 HTTP-date일 수 있습니다. 실습은 숫자 seconds만 사용합니다. production parser는 두 형식과 clock 차이를 다뤄야 합니다.

11.3. 429와 503의 차이

status	의미	화면 문장	retry
429	이 actor·resource의 요청이 너무 많음	제한·남은 대기	policy 안에서 가능
503	service가 일시적으로 처리 불가	서비스 상태·대기	operation 안전성 확인 후

11.4. 하지 않을 것

- disabled button 없이 countdown만 표시
- 0초 간격 무한 retry
- 409·422까지 자동 retry
- POST에 새 idempotency key를 매번 생성해 retry
- 사용자가 멈출 방법 없는 background loop

12. POST 재시도는 Idempotency Key로 보호합니다

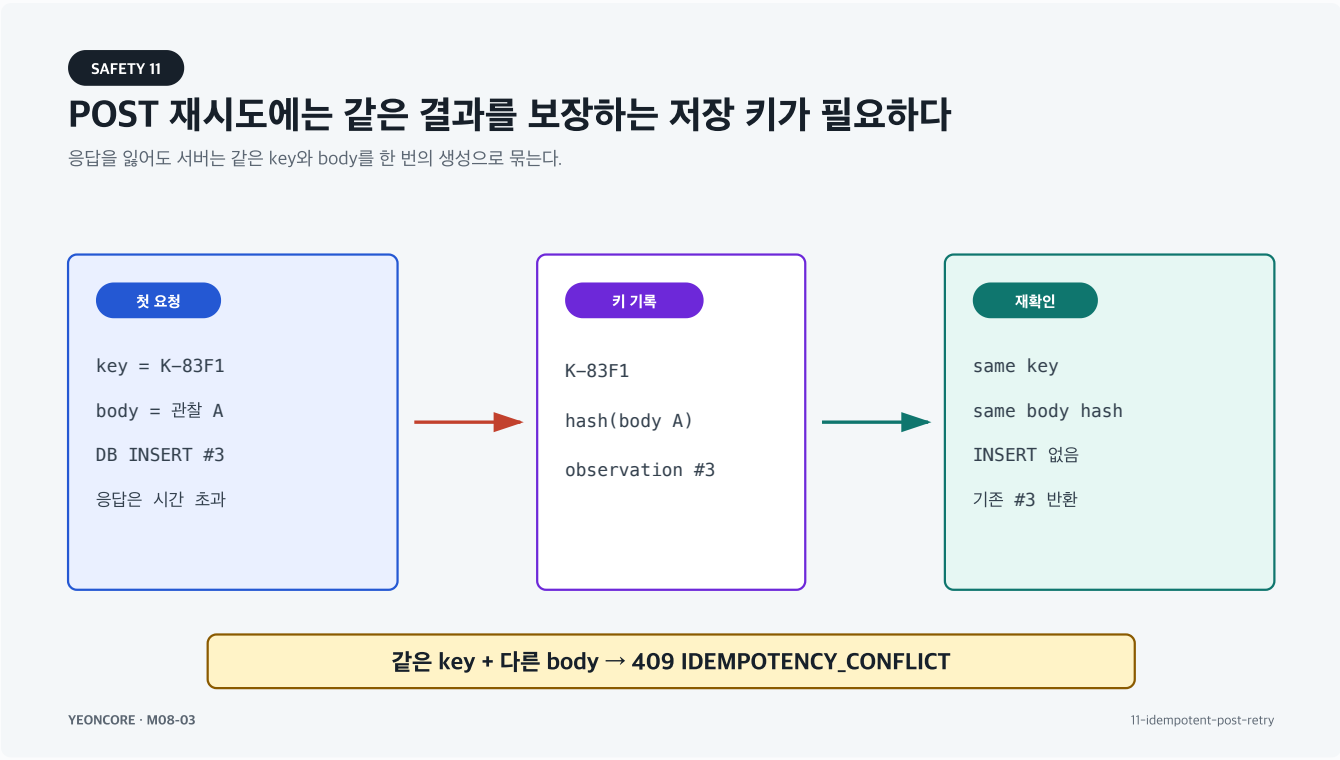
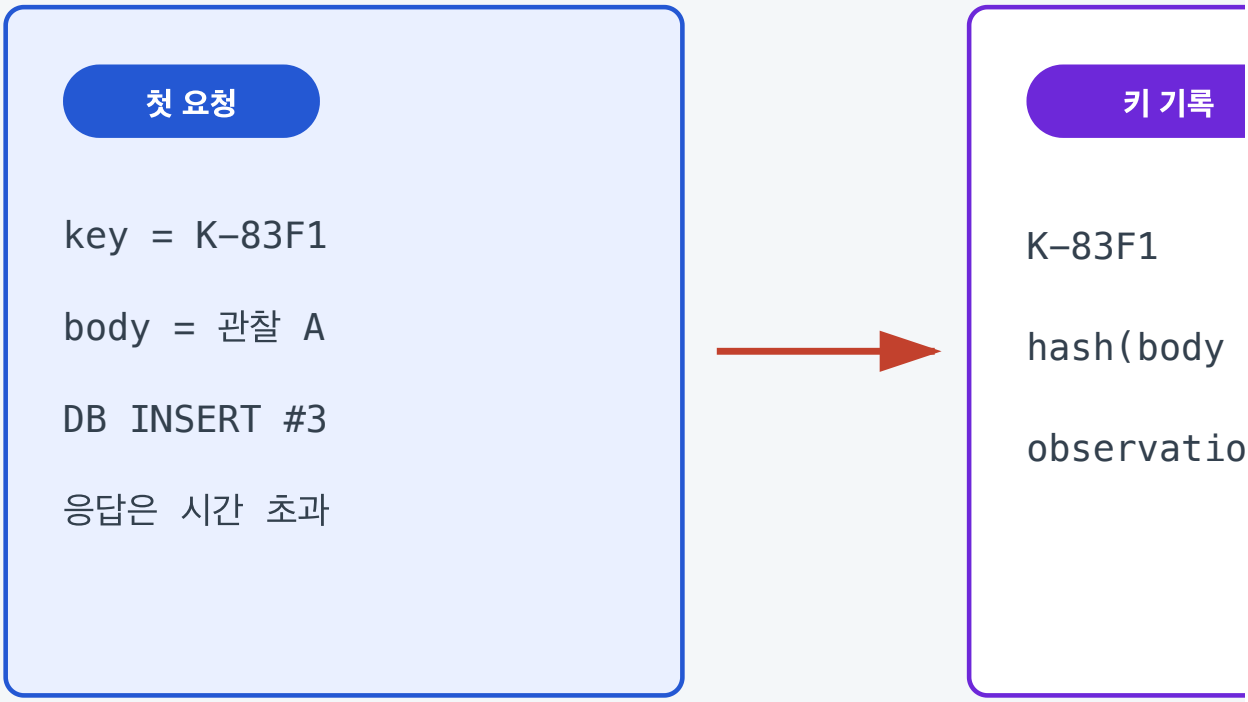
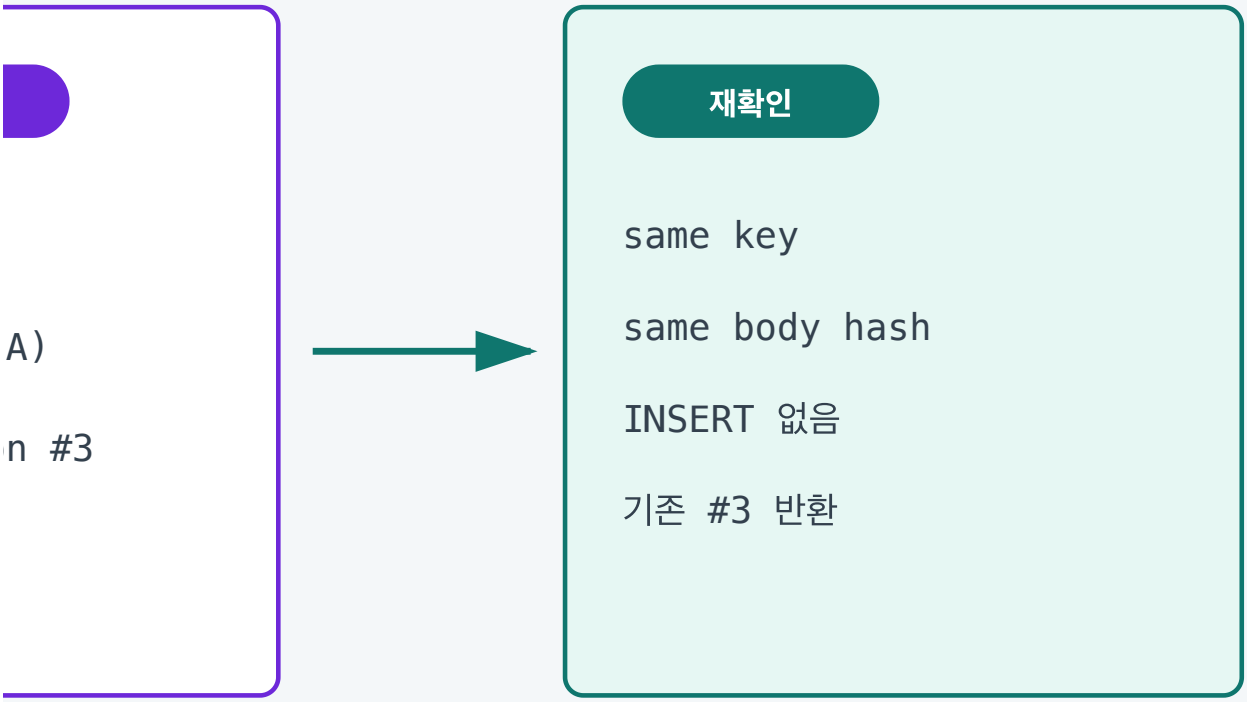


그림 12. 첫 요청이 commit 뒤 응답을 잃어도 같은 key와 같은 body는 기존 row를 반환합니다. 새 row를 만들지 않습니다.

응답을 잃어도 서버는 같은 key와 body를 한 번의 생성으로 묶는다.




A)
n #3
재확인
same key
same body hash
INSERT 없음
기존 #3 반환

12.1. Client contract

```
const pending = {
  body: { category, summary },
  key: crypto.randomUUID(),
};

await fetch("/api/observations", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    "X-CSRF-Token": csrfToken,
    "Idempotency-Key": pending.key,
  },
  body: JSON.stringify(pending.body),
});
```

12.2. Server contract

```
key 없음·형식 오류 → 400 IDEMPOTENCY_KEY_REQUIRED
처음 보는 key      → body hash + row를 한 transaction에 기록
같은 key + 같은 hash → 기존 row 반환, replayed=true
같은 key + 다른 hash → 409 IDEMPOTENCY_CONFLICT
```

12.3. Table

```
CREATE TABLE idempotency_keys (
  key TEXT PRIMARY KEY,
  user_id INTEGER NOT NULL REFERENCES users(id),
  request_hash BLOB NOT NULL,
  observation_id INTEGER NOT NULL REFERENCES observations(id),
  created_at TEXT NOT NULL
);
```

12.4. 보존 정책이 필요합니다

production에서는 key를 영원히 보관하지 않습니다. operation의 최대 재시도 기간, storage 비용, 개인정보·audit 요구를 고려해 TTL과 cleanup을 정합니다.

13. Timeout·Cancel·Stale Response를 분리합니다

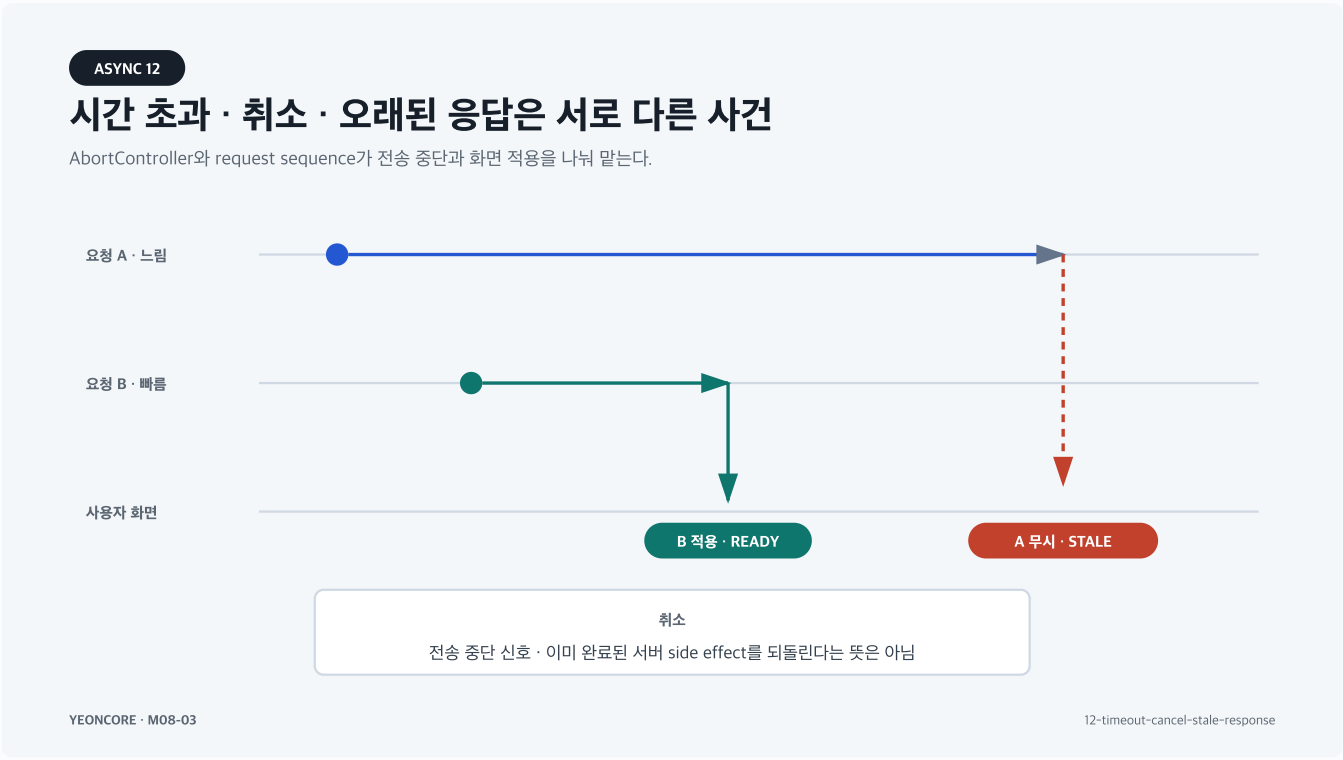
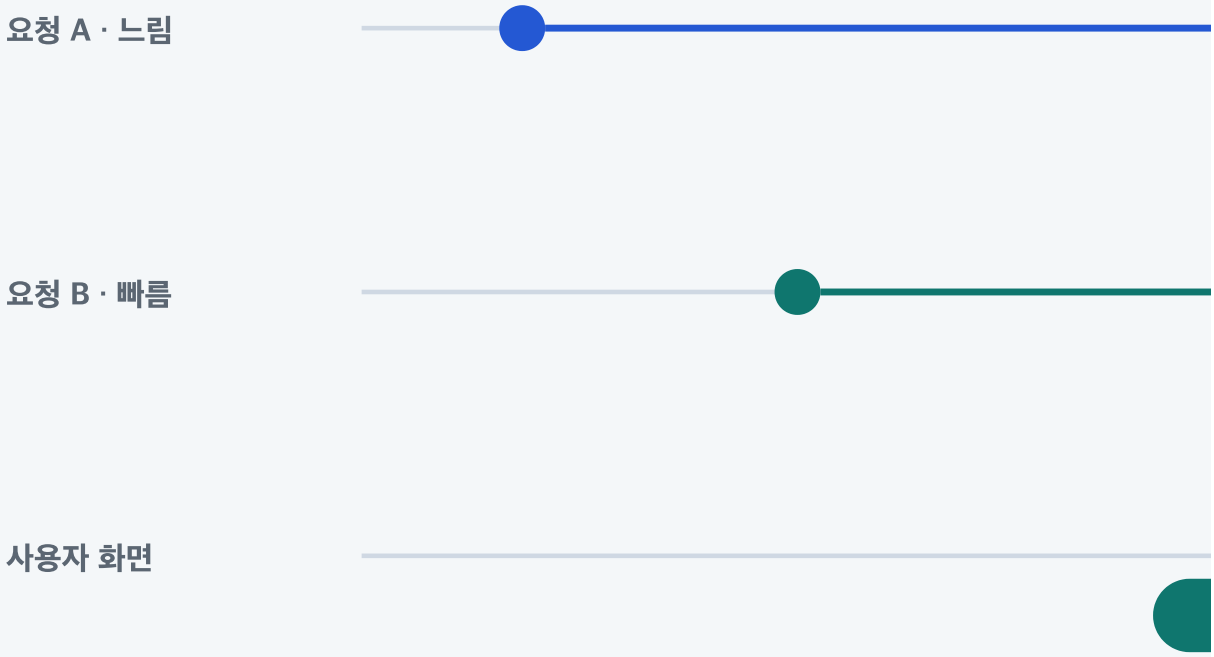


그림 13. 전송을 취소하는 일과 서버 작업을 취소하는 일은 같지 않습니다. 화면에는 가장 최근 의도의 응답만 적용합니다.

AbortController와 request sequence가 전송 중단과 화면 적용을 나눠 맡는다.



취

전송 중단 신호 · 이미 완료된 서버 s

소

side effect를 되돌린다는 뜻은 아님

13.1. AbortController

```
const controller = new AbortController();
const timer = setTimeout(() => controller.abort(), 500);

try {
  await fetch(url, { signal: controller.signal });
} finally {
  clearTimeout(timer);
}
```

browser가 fetch를 중단해도 server가 이미 request body를 받고 DB를 commit했을 수 있습니다. 그래서 POST timeout은 “실패”가 아니라 **결과 미확인**일 수 있습니다.

13.2. Stale response gate

```
const requestId = ++state.latestListRequest;
const payload = await requestJson("/api/observations");

if (requestId !== state.latestListRequest) {
  return "STALE_IGNORED";
}

state.observations = payload.data;
```

사용자가 filter A 뒤 filter B를 빠르게 선택하면 A 응답이 나중에 올 수 있습니다. arrival order가 아니라 user intent order를 적용합니다.

13.3. `navigator.onLine`은 힌트입니다

브라우저와 운영체제는 서로 다른 heuristic으로 online 여부를 판단합니다. LAN 연결은 있지만 실제 인터넷은 안 될 수 있습니다. `navigator.onLine`만으로 저장 버튼을 막지 않고 안내 신호로만 사용합니다.

14. 결과 미확인을 별도 상태로 표시합니다

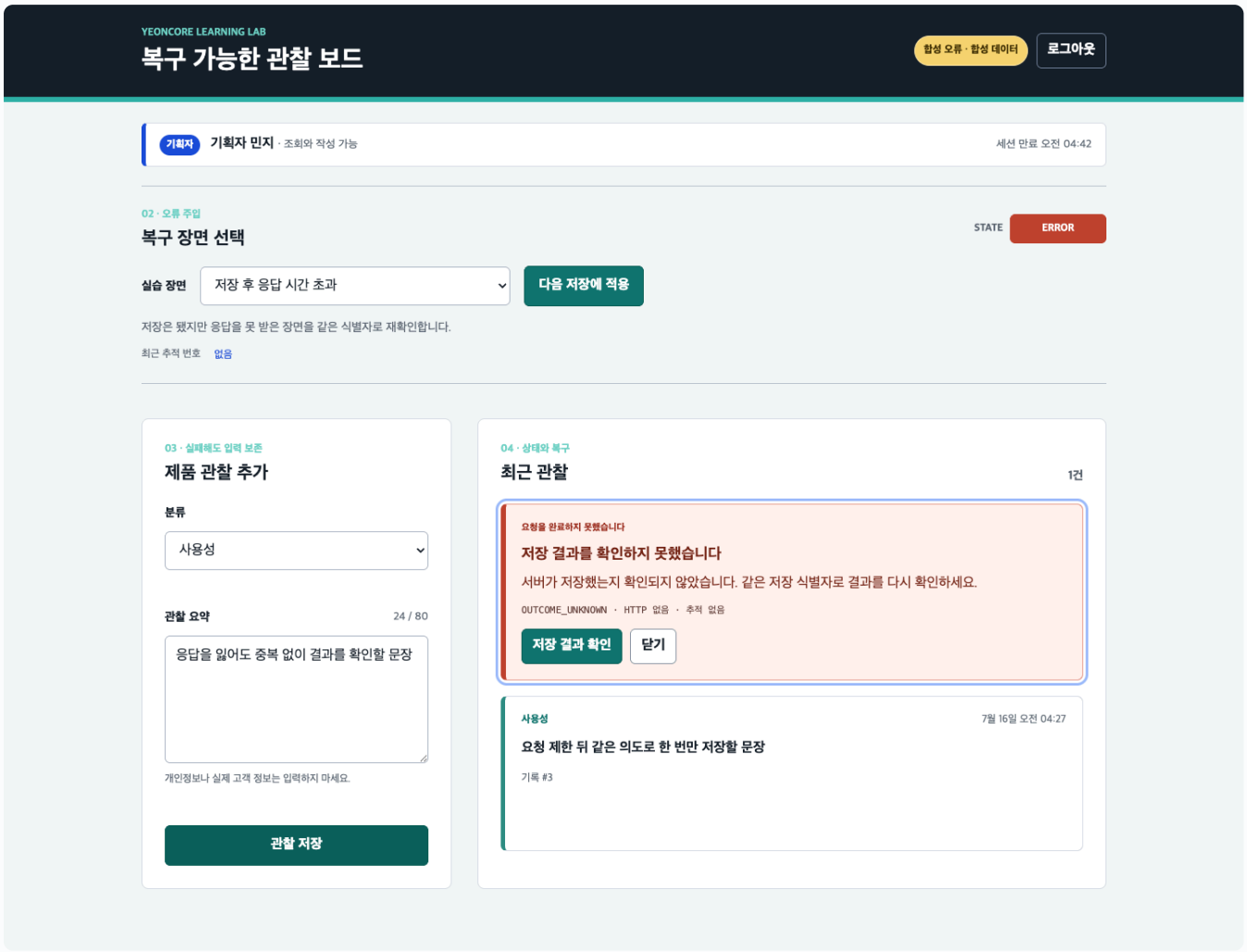


그림 14. timeout 순간에는 새 row가 목록에 아직 보이지 않지만 입력은 남습니다. `저장 결과 확인`은 같은 idempotency key를 재사용해 기존 row를 확인합니다.

14.1. 세 문장을 구분합니다

```
저장 실패: 서버가 적용하지 않았다는 증거가 있음
저장 성공: 성공 응답과 결과가 확인됨
결과 미확인: commit 여부를 알 수 없음
```

14.2. 실습의 증거

```
timeout 전 목록 = 3건
첫 POST = server row #4 commit, browser response abort
화면 = OUTCOME_UNKNOWN, draft 유지
같은 key로 결과 확인 = 201, replayed=true
확인 뒤 목록 = 4건
DB row delta = +1
```

15. 접근 가능한 오류와 상태를 만듭니다

15.1. 정상 진행은 polite status

```
<div role="status" aria-live="polite" aria-atomic="true"></div>
```

loading 완료, 저장 성공, 취소 같은 상태 메시지는 focus를 빼앗지 않고 보조 기술에 전달합니다.

15.2. 사용자가 즉시 알아야 할 오류는 alert

```
<section role="alert" tabindex="-1" aria-labelledby="error-title">
  <h3 id="error-title">입력값을 확인해 주세요</h3>
  <p>표시된 항목을 수정하세요.</p>
</section>
```

submit 실패 뒤 오류 요약에 focus를 옮기고, field에는 `aria-invalid`와 설명 연결을 제공합니다. 색만으로 오류를 전달하지 않습니다.

15.3. Focus는 예측 가능해야 합니다

사건	focus
정상 status update	현재 control 유지
submit 오류 요약	오류 summary
field error 확인	첫 invalid field로 이동 가능

사건	focus
retry countdown	button disabled, focus 강제 이동 없음
retry 성공	다음 자연스러운 input 또는 결과 heading

16. 사용자 메시지와 운영 로그를 분리합니다

16.1. 사용자에게 보여 줄 것

```
무엇이 안 되었는가
입력과 기존 데이터가 남았는가
지금 할 수 있는 다음 행동
지원에 전달할 trace ID
```

16.2. 사용자에게 보여 주지 않을 것

- stack trace·source path
- SQL·table·column 내부 정보
- session cookie·CSRF token·access token
- 비밀번호·요청 전체 body
- dependency version과 server banner
- 다른 사용자의 resource 존재 여부

16.3. Log event

```
timestamp · service · version · trace_id
method · route template · status · stable code
authenticated subject ID 또는 최소화된 actor reference
duration · retry/replay flag · sanitized context
```

하나의 interaction identifier로 browser에 표시한 trace와 server log를 연결합니다. 로그 자체의 실패도 app 전체를 무너뜨리지 않게 테스트합니다.

17. Fault Injection으로 실패와 복구를 증명합니다

TEST 13

오류는 기다리지 말고 주입해서 증명한다

각 장면마다 status, 화면 state, 보존, 복구, 중복 여부를 함께 기록한다.

장면	HTTP	UI STATE	핵심 증거
EMPTY	200	empty	목록 0건
FIELD	422	field_error	draft 유지
CONFLICT	409	conflict	최신본 확인
LIMIT	429	rate_limited	3초 대기
OUTCOME	없음	error	same key 확인

PASS는 오류가 사라졌다는 뜻이 아니라, 약속한 실패와 복구가 재현됐다는 뜻이다.

YEONCORE · M08-03

13-fault-injection-evidence-matrix

그림 15. PASS는 오류가 발생하지 않았다는 뜻이 아니라, 약속한 오류·화면 state·보존·복구가 정확히 재현되었다는 뜻입니다.

각 장면마다 status, 화면 state, 보존, 복구, 중복 여부를 함께 기록한다.

장면	HTTP	UI STATE
EMPTY	200	empty
FIELD	422	field_error
CONFLICT	409	conflict
LIMIT	429	rate_limited
OUTCOME	없음	error

PASS는 오류가 사라졌다는 뜻이 아니라

	핵심 증거
	목록 0건
	draft 유지
	최신본 확인
	3초 대기
	same key 확인

양소하 실패와 보고가 재현됐다는 뜻이다

17.1. 실습 fault header

```
X-Lab-Fault: empty
X-Lab-Fault: slow
X-Lab-Fault: validation
X-Lab-Fault: conflict
X-Lab-Fault: rate-limit
X-Lab-Fault: unavailable
X-Lab-Fault: malformed
X-Lab-Fault: timeout-after-commit
```

17.2. 증거 matrix

fault	expected HTTP	expected UI	preservation	recovery
`empty`	200	empty	-	first action
`slow` + cancel	없음	canceled	old data	manual reload
client network	없음	error	old data	retry
`validation`	422	field_error	draft + old data	edit
`conflict`	409	conflict	draft + old data	reload
`rate-limit`	429	rate_limited	draft	wait 3s
`unavailable`	503	error	draft	bounded retry
`malformed`	502	error	old data	safe fallback
`timeout-after-commit`	response 없음	error/outcome unknown	draft	same-key confirm

17.3. Negative evidence

```
422 뒤 form.reset() 없음
409 뒤 자동 POST 없음
429 대기 중 button enabled 아님
stale response가 list·trace를 덮지 않음
same key replay 뒤 row delta +1 초과 아님
500 detail에 stack·SQL·path 없음
```

18. 실습 앱의 Server Gate를 읽습니다

18.1. Protected write 순서

```
session → CSRF → permission → JSON·validation
→ lab fault → idempotency key → transaction → response
```

권한 없는 actor는 input parsing·idempotency table·DB insert 전에 종료합니다.

18.2. Problem response

```
payload = make_problem(
    status,
    code,
    detail,
    trace_id,
    errors=fields_to_errors(fields),
    retryable=retryable,
    retry_after_seconds=retry_after_seconds,
)
send(payload, content_type="application/problem+json")
```

18.3. 같은 key의 transaction

```
BEGIN
SELECT idempotency key
없음 → INSERT observation → INSERT idempotency record → COMMIT
있음·hash 일치 → 기존 observation SELECT
있음·hash 불일치 → 409
```

observation과 key record가 같은 transaction에 있어야 한쪽만 남는 불완전 상태를 줄입니다.

19. 실습 앱의 Browser Gate를 읽습니다

19.1. 안전한 response parser

```
const contentType = response.headers.get("Content-Type") || "";
const payload = contentType.includes("json")
  ? await response.json().catch(() => null)
  : null;

if (!response.ok && !payload?.code) {
  throw new RequestFailure(
    "서버가 약속한 오류 형식으로 응답하지 않았습니다.",
    { code: "INVALID_RESPONSE", kind: "protocol" },
  );
}
```

19.2. 안전한 DOM

```
errorTitle.textContent = error.title;
errorDetail.textContent = error.message;
```

server message를 `innerHTML`로 넣지 않습니다.

19.3. Retry action은 error 종류에서 결정합니다

```
if (error.status === 409) retryAction = loadLatest;
else if (error.retryable) retryAction = retrySameOperation;
else retryAction = null;
```

20. 생성기와 Evidence Packet을 실행합니다

20.1. 생성

```
./02_Labs/G08_Fullstack/L08-03_create-resilient-error-state-practice.sh
```

20.2. 생성되는 evidence

```
evidence/
├── 01-reset.txt
├── 02-tests.txt
├── 03-resilience-audit.txt
├── 04-fault-contract-probe.json
├── 05-state-contract.json
└── 06-versions.txt
```

20.3. 자동 검사 기준

```
40 tests PASS
25/25 resilience audit PASS
fault contract probe PASS
Python 3.12 + 3.14 compatibility PASS
anonymous 401 · viewer write 403
422 · 409 · 429 · 503 distinct
same-key first 201 · replay 201 · row delta 1
```

20.4. 실행

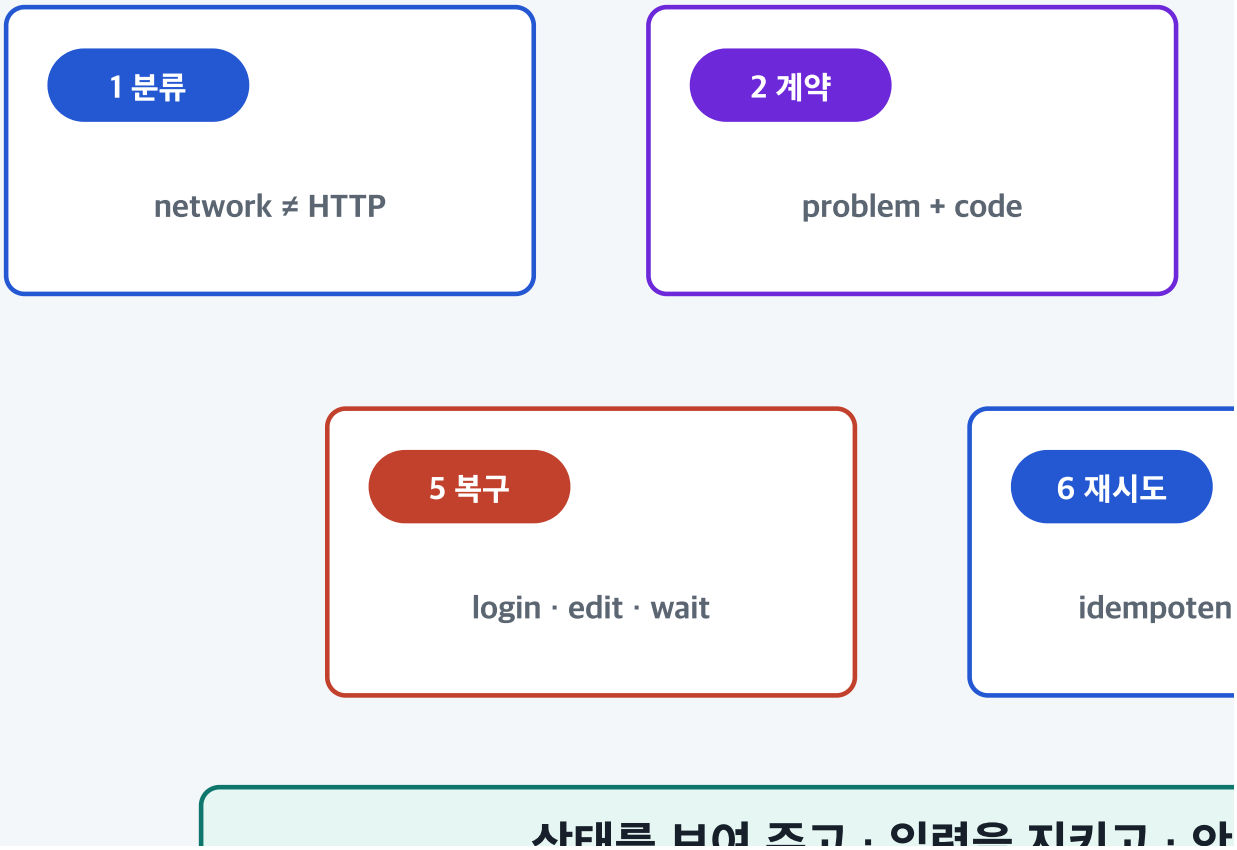
```
cd gibalja-resilient-error-state-practice/app
python3 app.py --host 127.0.0.1 --port 4173
```

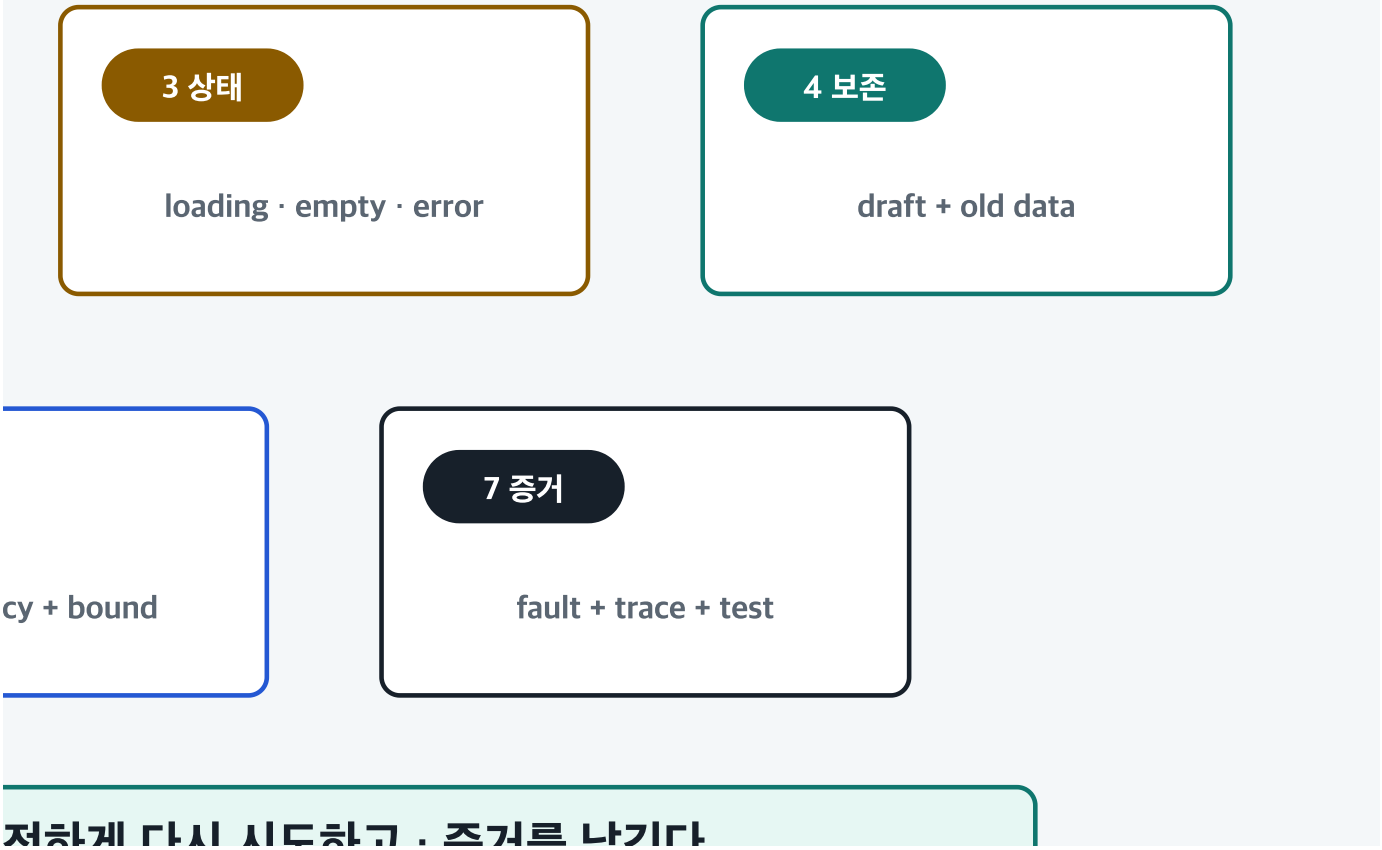
브라우저에서 `http://127.0.0.1:4173`을 엽니다.

21. 완료를 Seven Gates로 판정합니다

그림 16. 오류 처리는 분류·계약·상태·보존·복구·재시도·증거의 일곱 문을 모두 통과해야 완료됩니다.

한 문이라도 비어 있으면 실제 서비스의 오류 처리가 완성되지 않았다.




3 상태
loading · empty · error
4 보존
draft + old data
cy + bound
7 증거
fault + trace + test

gate	PASS 질문	evidence
1 분류	network·abort·HTTP·protocol을 나눴는가	decision table
2 계약	status·problem·code가 안정적인가	response sample
3 상태	loading·empty·ready·error가 보이는가	screenshot
4 보존	submit 실패 뒤 draft·기존 data가 남는가	before/after
5 복구	401·403·409·422·429·503의 next action이 다른가	recovery matrix
6 재시도	POST 중복과 무한 retry를 막는가	same-key row delta
7 증거	fault·trace·test·reset이 재현되는가	evidence packet

```
7/7 PASS       → READY FOR LEARNER PILOT
1개 UNKNOWN    → NOT READY, evidence 필요
1개 BLOCK      → NOT READY, contract 수정
```

22. 실무 제출 Packet

22.1. 필수 파일

- [ ] 오류·상태·복구 matrix
- [ ] 성공·실패 Problem Details sample
- [ ] list·submit state machine
- [ ] 422 field error screenshot
- [ ] 409·429·503 recovery screenshot 또는 trace
- [ ] timeout outcome-unknown record
- [ ] idempotency first·replay·row delta evidence
- [ ] accessibility keyboard·screen reader check
- [ ] sanitized log·trace sample
- [ ] version·environment·reset·known limitation

22.2. 한계 문장 예시

```
이 packet은 loopback 단일 process·SQLite·합성 data에서
오류 계약과 UI 복구를 증명한다.
실제 load balancer, distributed idempotency, central tracing,
multi-region failure, production rate limit과 개인정보 보존은 증명하지 않는다.
```

23. 셀프 테스트 · 먼저 답을 가리고 풉니다

문제 1

`fetch()`가 resolve되었고 HTTP 503이 돌아왔습니다. 통신 성공입니까, 업무 성공입니까?

정답 보기
통신에서는 Response를 받았지만 업무 성공은 아닙니다. `response.ok`는 false이며 503 Problem Details와 retry 조건을 읽어야 합니다.

문제 2

빈 목록 응답 `200 {"data":[]}`은 error state입니까?

정답 보기
아닙니다. 요청은 성공했고 결과가 0건인 `empty` state입니다. 첫 행동을 안내해야 합니다.

문제 3

Problem Details의 `detail` 문장을 화면 분기 key로 써도 될까요?

정답 보기
안 됩니다. 번역·문장 개선으로 바뀔 수 있습니다. 안정적인 `code`, HTTP status, typed extension을 사용합니다.

문제 4

422 뒤 form을 reset하면 어떤 계약이 깨집니까?

정답 보기
입력 보존과 수정 가능성입니다. 사용자가 잘못된 필드만 고칠 수 있도록 draft를 유지하고 field message를 연결해야 합니다.

문제 5

401과 403 모두 로그인 화면으로 보내면 무엇이 문제입니까?

정답 보기
403 사용자는 이미 인증되어 있습니다. 같은 role로 다시 로그인해도 permission이 바뀌지 않으므로 허용 행동·권한 요청 같은 인가 복구가 필요합니다.

문제 6

409을 받은 POST를 즉시 같은 조건으로 자동 반복해도 될까요?

정답 보기
아닙니다. 최신 resource state를 읽고 차이를 해결한 뒤 새 조건으로 제출해야 합니다.

문제 7

429 `Retry-After: 3`을 받았습니다. 버튼은 어떻게 보여야 합니까?

정답 보기
최소 3초 동안 disabled 상태와 남은 대기를 알리고, 시간이 지난 뒤 사용자가 수동 재시도할 수 있게 합니다.

문제 8

POST timeout 직후 새 idempotency key로 다시 보내면 어떤 위험이 있습니까?

정답 보기
첫 요청이 이미 commit되었다면 두 번째 row가 생성될 수 있습니다. 같은 operation의 같은 key와 body로 결과를 재확인해야 합니다.

문제 9

AbortController로 fetch를 취소하면 server transaction도 자동 rollback됩니까?

정답 보기
보장되지 않습니다. server가 이미 처리·commit했을 수 있습니다. 취소와 server-side effect cancellation을 구분합니다.

문제 10

느린 filter A 응답이 빠른 filter B 뒤에 도착했습니다. 무엇을 적용합니까?

정답 보기
가장 최근 사용자 의도인 B만 적용합니다. request sequence를 비교해 A를 stale response로 무시합니다.

문제 11

`navigator.onLine === false` 이면 저장 기능을 영구 disable해야 합니까?

정답 보기
아닙니다. 값은 환경별 heuristic이라 신뢰할 수 없는 경우가 있습니다. offline 가능성을 알리는 힌트로 사용하고 실제 요청 결과로 판단합니다.

문제 12

오류 처리 완료를 무엇으로 증명합니까?

정답 보기
정상 화면 한 장이 아니라 fault별 HTTP·code·UI state·입력·기존 data 보존·복구 행동·trace·DB row delta·reset을 묶은 evidence packet으로 증명합니다.

24. 공식 출처와 적용 원칙

출처	이 교재에 적용한 내용
RFC 9457 Problem Details for HTTP APIs	`type` · `title` · `status` · `detail` · `instance`, extension, 보안 고려
RFC 9110 HTTP Semantics	401·403·409·422·503, Retry-After, safe·idempotent method·retry 의미
RFC 6585 Additional HTTP Status Codes	429 Too Many Requests와 Retry-After
WHATWG Fetch Standard	network error·aborted network error·response model
MDN Response.ok	200~299 성공 여부 판정
MDN AbortController	fetch·response body·stream 중단 signal
MDN Navigator.onLine	online heuristic를 기능 차단이 아닌 힌트로 사용
WCAG 2.2	error identification·suggestion·status message·focus order
WAI ARIA22 role=status	polite live status와 atomic announcement
OWASP Error Handling Cheat Sheet	내부 구현 정보 노출 방지·일관된 오류 처리
OWASP Logging Cheat Sheet	interaction identifier·event attribute·민감 data 제외·logging failure test

출처를 읽는 방법

1. HTTP status는 app-specific message를 대신하지 않지만 의미를 다시 정의하지도 않습니다.
2. Problem Details는 debug stack을 사용자에게 보내는 형식이 아닙니다.
3. retry 가능성은 status만으로 결정하지 않고 operation semantics·commit 가능성·idempotency를 함께 봅니다.
4. 접근성 technique은 실제 browser·보조 기술 조합에서 검증합니다.
5. 실습의 합성 fault와 production 장애 대응을 분리해 기록합니다.

다음 매뉴얼: M09-01 AI 서비스는 어떻게 움직이는가