M08-01 · GIBALJA VISUAL MANUAL

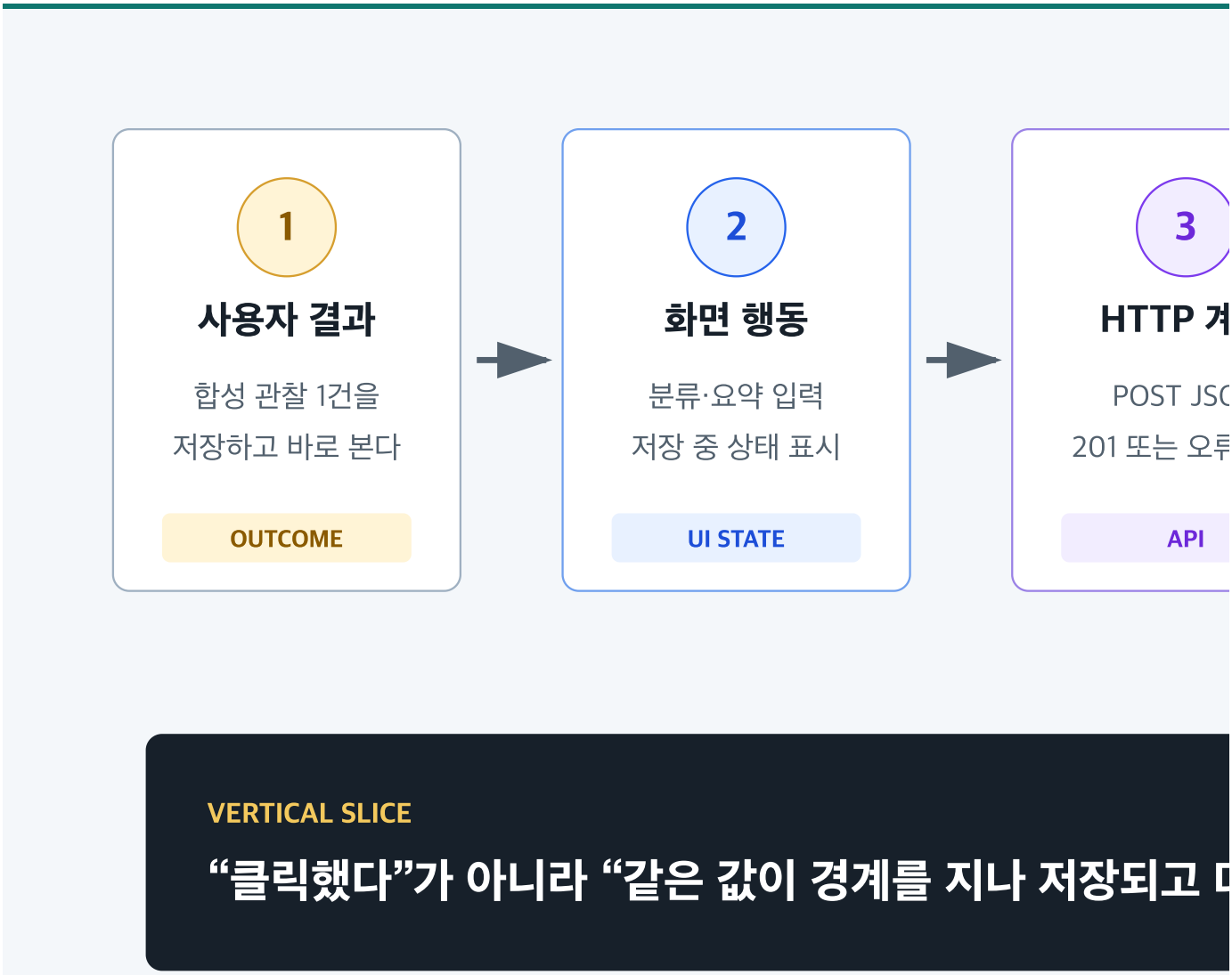
화면·API·DB를 연결한 MVP 만들기

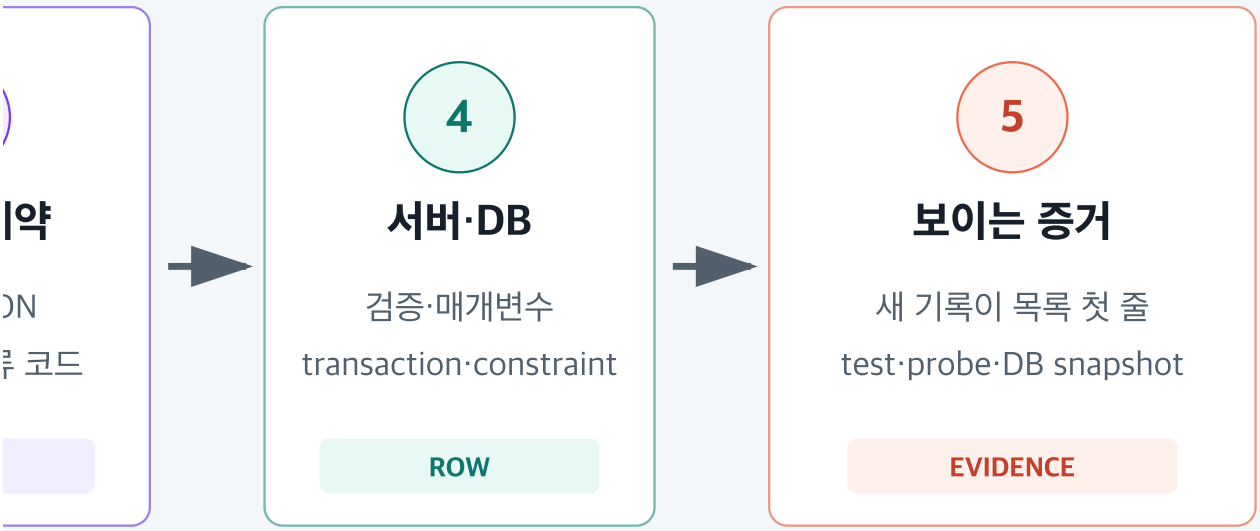
> **한 문장 목표:** `사용자 결과 → 화면 상태 → HTTP 계약 → 서버 검증 → 매개변수화 SQL → SQLite transaction → JSON 응답 → 다시 보이는 화면 → 자동 증거`를 한 줄로 연결해, 작은 기능 하나를 처음부터 끝까지 재현합니다.

난이도	개념	실습	셀프 테스트	최종 산출물
Level 1	60분	60분	15분	실행 가능한 관찰 보드, 세 계약 지도, HTTP·DB 증거, reset packet

풀스택을 공부할 때 HTML·API·SQL을 따로 외우면 파일은 늘어나지만 서비스가 연결되지 않습니다. 이번에는 **합성 관찰 1건을 저장하고 곧바로 보는 결과**를 기준으로 모든 층을 한 번 왕복합니다. 실제 개인정보·외부 API·로그인·배포는 의도적으로 제외합니다.

그림 1. 첫 MVP의 크기는 파일 수가 아니라 한 사용자 결과가 모든 경계를 통과해 증거로 닫히는 범위로 정합니다.


1
사용자 결과
합성 관찰 1건을
저장하고 바로 본다
OUTCOME
2
화면 행동
분류·요약 입력
저장 중 상태 표시
UI STATE
3
HTTP 계
POST JSO
201 또는 오류
API
VERTICAL SLICE
“클릭했다”가 아니라 “같은 값이 경계를 지나 저장되고 다



다시 보였다”까지 증명합니다.

0. 이 PDF를 공부하는 방법

1회차 · 그림만 읽기 · 20분

그림 1부터 그림 15까지 제목과 결론 띠만 읽습니다. 다음 일곱 문장을 말할 수 있으면 됩니다.

```
MVP는 작은 화면이 아니라 작은 사용자 결과다.
한 필드는 UI·API·DB에서 같은 뜻과 규칙을 가져야 한다.
브라우저 검증은 편의이고 서버 검증은 신뢰 경계다.
SQL 구조와 사용자 값은 placeholder로 분리한다.
transaction은 모두 반영하거나 아무것도 남기지 않는다.
화면은 loading·empty·ready·error 상태를 모두 가진다.
완료는 start·test·reset·evidence를 다른 사람이 재현할 때다.
```

2회차 · 실습 묶음 생성 · 10분

풀스택 MVP 실습 생성기를 실행합니다.

```
./02_Labs/G08_Fullstack/L08-01_create-fullstack-mvp-practice.sh
```

생성기는 외부 package와 network 없이 다음을 만듭니다.

```
gibalja-fullstack-mvp-practice/
├── app/       실행 가능한 UI·API·SQLite MVP
└── evidence/  reset·test·audit·HTTP probe·DB snapshot·version
```

이미 같은 target이 있으면 exit code `2`로 멈추며 덮어쓰지 않습니다.

풀스택 연결 판독 데스크에서 12개 장면을 먼저 풀면 어느 경계에서 어떤 증거를 찾아야 하는지 빠르게 예습할 수 있습니다.

3회차 · 왕복 추적 · 35분

브라우저 저장 버튼을 누르기 전에 다음 빈칸을 채웁니다.

```
사용자 outcome =
UI input / state =
HTTP method / path / body =
server validation =
SQL / transaction =
DB row =
response status / body =
다시 보이는 화면 =
failure evidence =
```

4회차 · 내 기능에 적용 · 70분

풀스택 MVP 연결 기록 양식을 채우고 단계별 실습서에 따라 수행합니다. 낯선 표현은 풀스택 MVP 연결 용어집에서 찾습니다.

1. MVP를 화면 수가 아니라 Vertical Slice로 정의합니다

`MVP`를 “기능이 적은 제품”이라고만 이해하면 중요한 실패 경로와 데이터 규칙을 빠뜨리기 쉽습니다. 이번 교재에서 MVP는 다음 질문에 답하는 최소 실험입니다.

```
한 actor가 한 행동을 했을 때,
가장 중요한 outcome이 실제 경계를 통과해 보이는가?
그리고 다른 사람이 같은 시작점에서 같은 결과를 재현할 수 있는가?
```

1.1. 수평 작업과 수직 작업

방식	범위	눈에 보이는 진척	연결 위험
수평 slice	화면 전체, API 전체, DB 전체 중 한 층	파일·endpoint·table이 많아짐	층 사이 계약이 늦게 충돌
수직 slice	사용자 행동 하나를 UI→API→DB→UI로 왕복	실제 결과 1개가 완결	작은 범위에서 즉시 통합 문제 발견

첫 slice는 작아야 하지만 얕아서는 안 됩니다. 정상 장면만 보여 주는 demo와 실패·초기화·증거가 있는 학습 MVP는 다릅니다.

1.2. 이번 slice의 actor와 outcome

```
actor: 제품 기획자
action: 합성 제품 관찰의 분류와 요약을 입력하고 저장
outcome: 생성된 id와 함께 새 관찰이 최근 목록 첫 줄에 나타남
success evidence:
  - POST /api/observations = 201
  - response.data.summary = trim된 입력
  - GET /api/observations meta.count 증가
  - SQLite row 증가
  - 화면 완료 알림과 새 목록 항목
```

30초 확인 1
"관찰 등록 화면 만들기"와 "관찰 1건이 저장되고 다시 보이기" 중 어느 쪽이 검증 가능한 outcome인가요? 후자의 증거를 화면·HTTP·DB에서 하나씩 적습니다.

2. Outcome과 Non-goal로 범위를 잠급니다

작은 slice가 자꾸 커지는 이유는 구현 능력이 부족해서가 아니라 제외 범위를 기록하지 않았기 때문입니다.

2.1. 이번에 포함하는 것

포함	이유
분류 3개와 요약 5~80자	작지만 의미 있는 업무 규칙
목록 조회·생성·단건 조회	저장 전후를 HTTP로 확인
server validation	브라우저 우회에 대비한 신뢰 경계
SQLite constraint·transaction	저장 무결성과 실패 원자성
loading·empty·ready·error	비동기 화면의 최소 상태 집합
reset·test·audit·probe	다음 사람이 처음부터 재현

2.2. 이번에 제외하는 것

```
로그인·인증·권한           → M08-02에서 추가
실제 고객·직원 정보        → 합성 data만 사용
외부 API·email·analytics  → network effect 없음
file upload               → binary·storage 위험 제외
pagination·search         → 최근 20건으로 고정
cloud·container·deployment→ local loopback만 사용
production server         → Python http.server는 학습용
```

Python 공식 문서는 `http.server`가 기본적인 보안 검사만 구현하므로 운영 환경에 권장되지 않는다고 명시합니다. 이번 server는 HTTP 경계를 눈으로 배우는 local 실습 장치입니다. 운영용 framework 선택·TLS·reverse proxy·process 관리로 범위를 확대하지 않습니다.

2.3. Stop condition

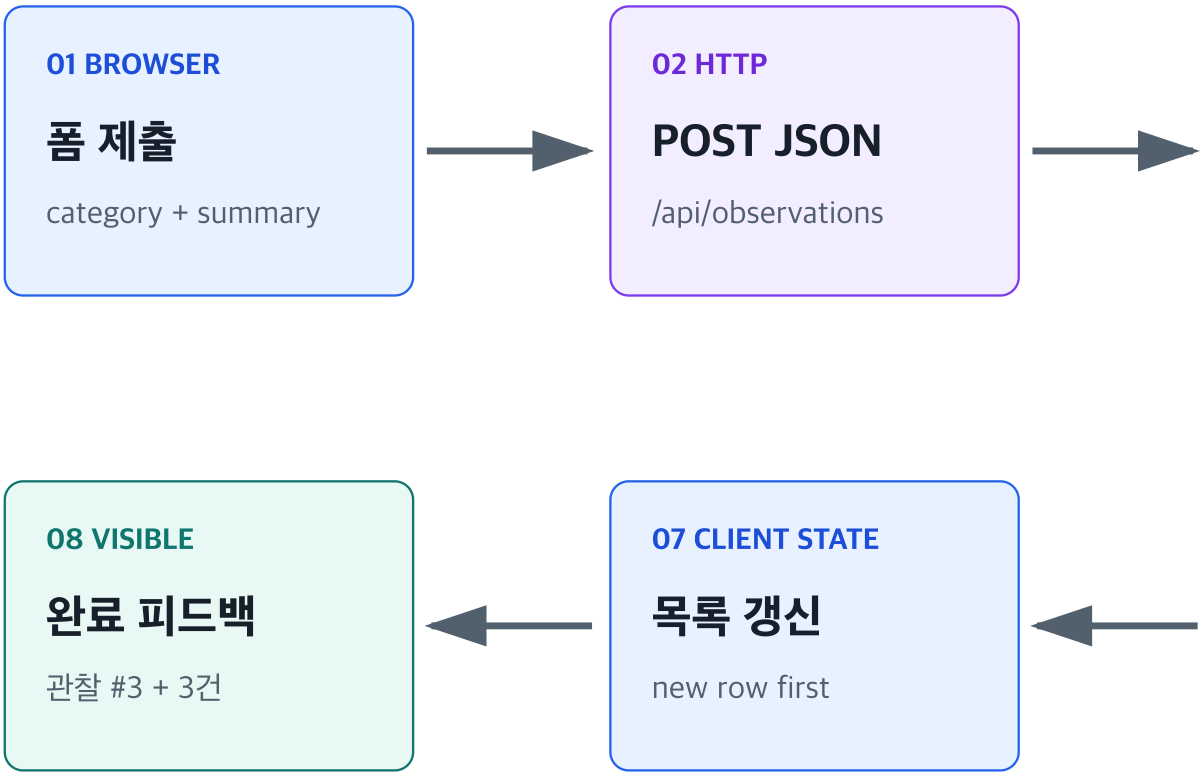
다음 상황이면 구현을 멈추고 scope를 다시 봅니다.

- 실제 개인정보가 필요해짐
- 로그인 없이 누가 쓸 수 있는지 판단해야 함
- 외부 system에 message를 보내야 함
- schema migration이나 운영 data 보존이 필요함
- “이왕이면”으로 endpoint가 세 개 이상 늘어남

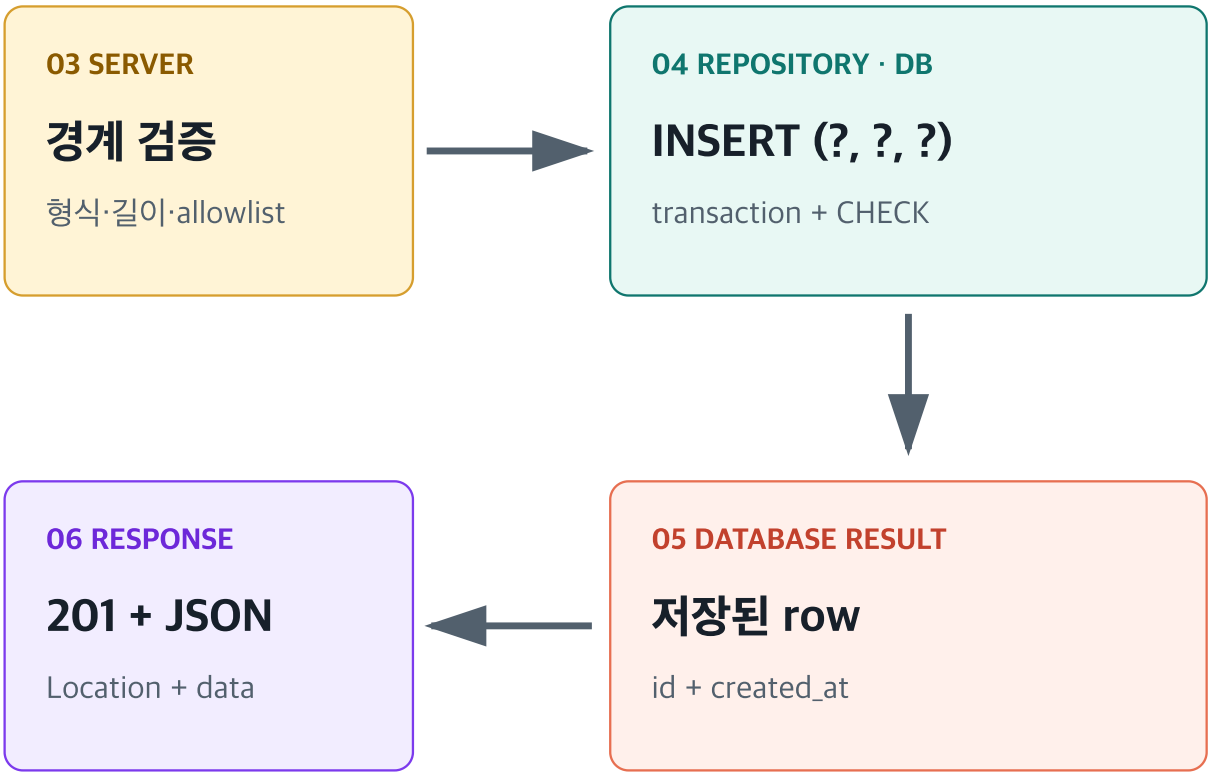
안전 경계
생성된 앱에는 합성 문장만 입력합니다. `reset_db.py`는 local 연습 DB를 삭제하고 다시 만듭니다. 실제 운영 database나 복사본에 이 절차를 적용하지 않습니다.

3. 한 번의 저장을 왕복 경로로 추적합니다

그림 2. 오류가 발생하면 마지막으로 확인된 장면과 다음 장면 사이를 조사합니다. 모든 층을 한꺼번에 의심하지 않습니다.



오류가 나면 마지막으로 확인된 장



면과 다음 장면 사이를 조사합니다.

3.1. Request-to-row

```
form value
→ JSON.stringify
→ HTTP request bytes
→ JSON parse
→ server validation
→ normalized value
→ DB-API bound value
→ SQLite row
```

각 화살표는 표현이 바뀌는 지점입니다. 화면의 `분류`는 request에서 `category`, DB에서도 `category`가 됩니다. 화면의 `사용성` label은 저장하지 않고 stable code `USABILITY`를 저장합니다.

3.2. Row-to-view

```
SQLite row: created_at
→ Python dict: createdAt
→ JSON text
→ browser object
→ client state.observations
→ DOM textContent
→ 사람이 보는 날짜·요약·기록 번호
```

DB의 `snake_case`와 JSON의 `camelCase`가 다르면 변환 책임을 한 곳에 둡니다. 실습에서는 repository의 `_serialize()`가 이 책임을 가집니다.

3.3. 왕복 trace key

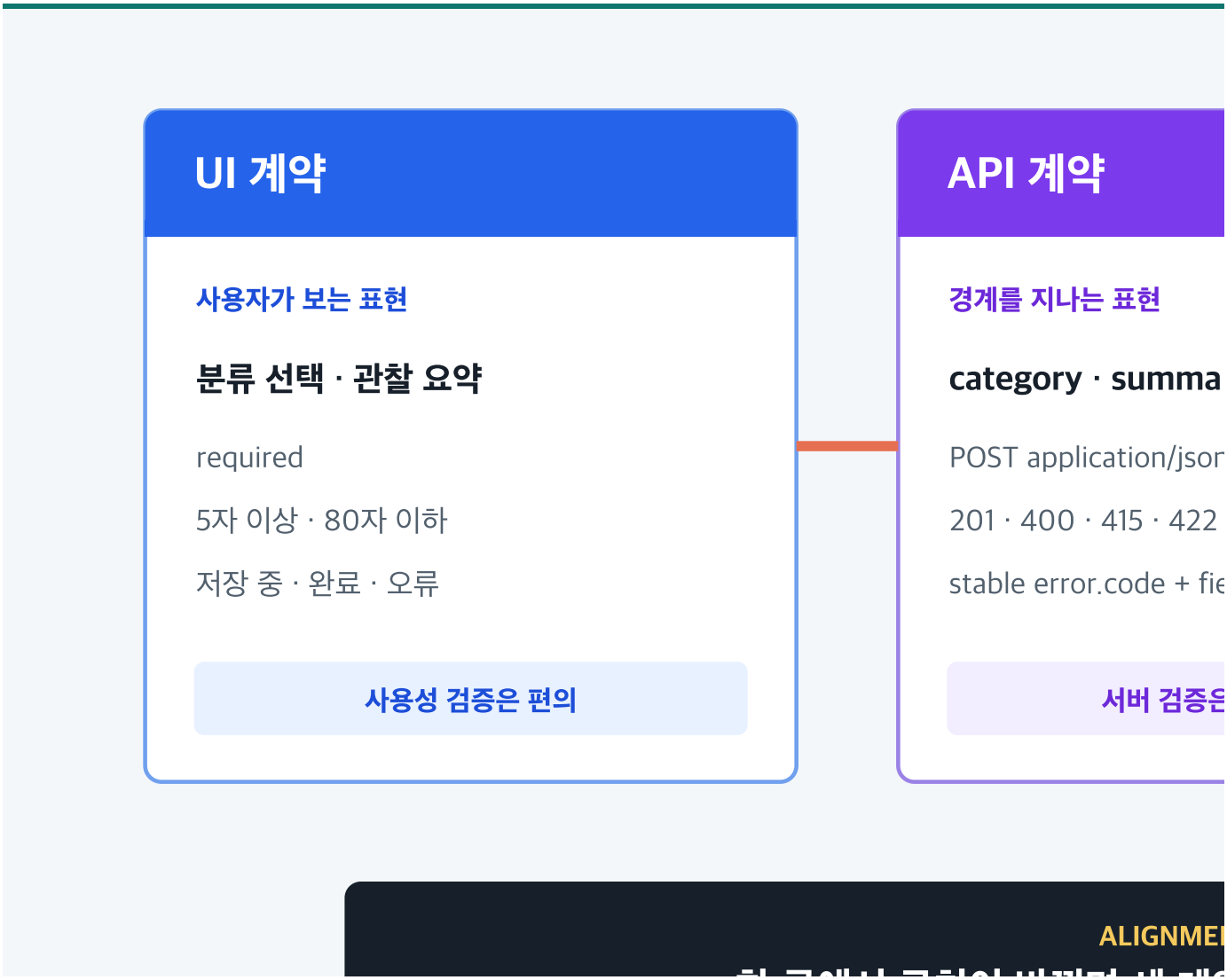
초보 실습에서는 생성된 정수 `id`가 가장 단순한 trace key입니다.

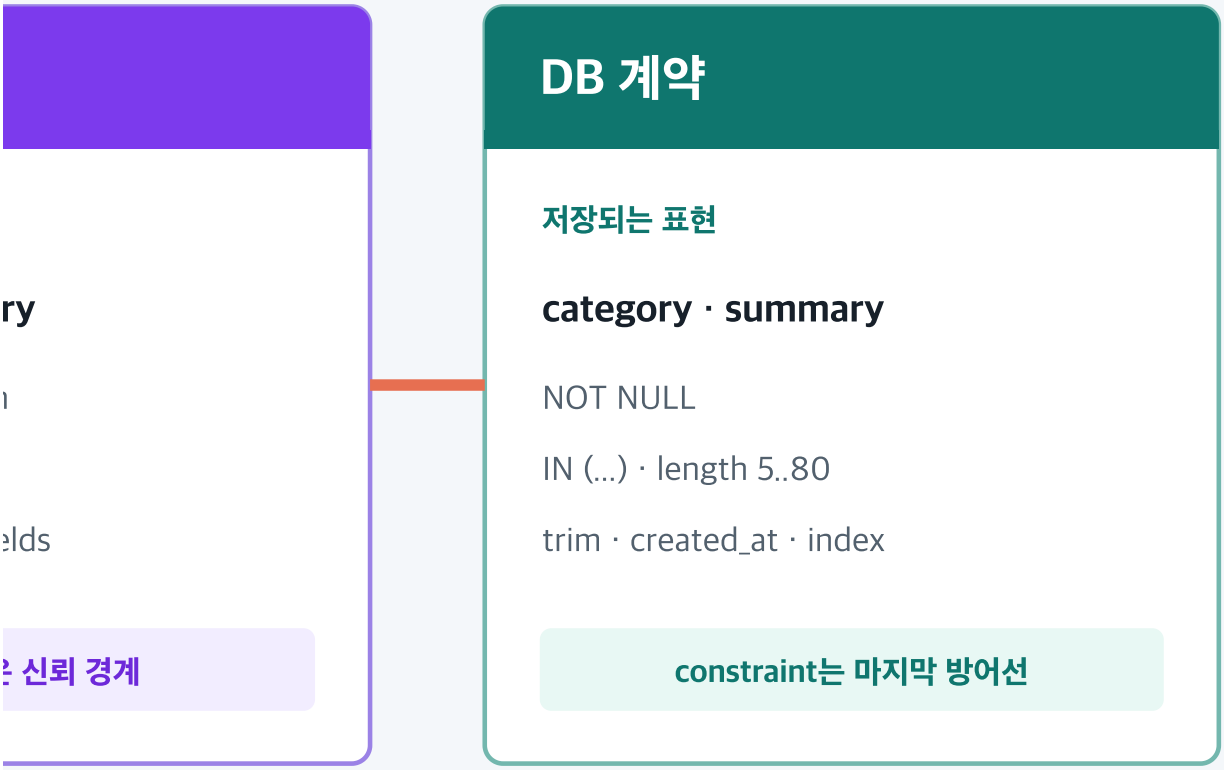
위치	trace key 예
DB	`observations.id = 3`
response	`data.id = 3`
header	`Location: /api/observations/3`
UI	`기록 #3`

위치	trace key 예
완료 알림	`관찰 #3 저장을 완료했습니다.`

4. UI·API·DB 세 계약을 한 줄로 정렬합니다

그림 3. 브라우저 검증, 서버 검증, DB 제약은 같은 규칙을 복사한 것이 아니라 서로 다른 신뢰 지점에서 같은 의미를 보존합니다.


UI 계약
사용자가 보는 표현
분류 선택 · 관찰 요약
required
5자 이상 · 80자 이하
저장 중 · 완료 · 오류
사용성 검증은 편의
API 계약
경계를 지나는 표현
category · summa
POST application/jsor
201 · 400 · 415 · 422
stable error.code + fie
서버 검증은
ALIGNME


ry
elds
신뢰 경계
DB 계약
저장되는 표현
category · summary
NOT NULL
IN (...) · length 5..80
trim · created_at · index
constraint는 마지막 방어선


NT CHECK

4.1. Field alignment table

의미	UI	API	DB	핵심 규칙
관찰 분류	`select#category`	`category` string	`category TEXT`	3개 stable code allowlist
관찰 요약	`textarea#summary`	`summary` string	`summary TEXT`	trim 뒤 5~80자
생성 번호	목록 `기록 #n`	`data.id` number	`id INTEGER PK`	server·DB 생성, client 입력 금지
생성 시각	사람용 날짜	`createdAt` ISO string	`created_at TEXT`	server 생성, client 입력 금지

4.2. Contract drift를 찾는 질문

```
UI가 허용하지만 API가 거부하는 값은 무엇인가?
API가 허용하지만 DB가 거부하는 값은 무엇인가?
DB에 있는데 response가 누락한 필드는 무엇인가?
response에 있지만 UI가 해석하지 못하는 값은 무엇인가?
한 곳의 이름이 바뀌면 어떤 test가 실패해야 하는가?
```

4.3. 안정 코드와 표시 문구

```
stable code: USABILITY   → API·DB·조건문
display label: 사용성     → 화면·번역
```

표시 문구는 바뀔 수 있지만 stable code는 계약 변경 없이 바꾸지 않습니다. 한국어 label을 DB key로 저장하면 번역·문구 수정이 data migration으로 번집니다.

5. POST 요청을 네 덩어리로 읽습니다

그림 4. method·path·headers·body를 분리해 읽은 뒤, server는 경로부터 업무 allowlist까지 순서대로 검증합니다.

REQUEST

```
POST /api/observations HTTP/1.1
Content-Type: application/json
Content-Length: 96

{
  "category" : "USABILITY"
  "summary" : "저장 완료 상태를 확인합니다."
}
```

UI의 required 속성을 최적화이

SERVER BOUNDARY

1 경로·메서드가 맞는가

2 JSON 형식과 UTF-8인가

3 본문이 4KB 이하인가

4 필드·형식·길이가 맞는가

5 업무 allowlist에 속하는가

5.1. Method와 path

```
POST /api/observations
```

`POST`는 이번 계약에서 새 resource 생성 의도를 뜻합니다. 다른 경로에 `POST`하면 `404`, 같은 경로에 `DELETE`하면 `405`와 `Allow: GET, POST`를 돌려줍니다.

5.2. Headers

header	이번 규칙	실패
`Content-Type`	`application/json`	`415 UNSUPPORTED_MEDIA_TYPE`
`Content-Length`	존재하고 0~4096 byte	`411`, `400`, `413`

본문 크기 제한은 JSON parse 전에 확인합니다. 큰 body를 먼저 전부 읽고 나서 거부하면 memory와 처리 시간을 이미 소비한 뒤입니다.

5.3. Body

```
{
  "category": "USABILITY",
  "summary": "저장 완료 상태를 확인합니다."
}
```

허용 필드는 정확히 두 개입니다. `role`, `id`, `createdAt` 같은 알 수 없는 필드를 조용히 무시하지 않고 `422 UNKNOWN_FIELD`로 거부합니다. client가 server 소유 필드를 주입하는 것을 막고 계약 drift를 빨리 발견하기 위해서입니다.

> **30초 확인 2**
> 브라우저 개발자 도구에서 request를 볼 때 URL과 JSON만 보지 말고, method·Content-Type·status·response error code까지 한 줄로 기록합니다.

6. Endpoint 계약에 성공과 실패를 함께 적습니다

그림 5. endpoint 계약은 path 목록이 아니라 성공 body·실패 code·불변식을 소비자와 합의하는 문서입니다.

METHOD · PATH	성공
GET /api/observations 최근 목록 최대 20건	**200** data[] + count
POST /api/observations 합성 관찰 1건 생성	**201** data + Location
GET /api/observations/{id} 저장 결과 단건 확인	**200** data object

2xx · 결과 전달

4xx · 입력·계약 수정

주요 실패	불변식
500 generic error	newest first no-store
400 JSON · 413 size 415 type · 422 value	unknown field 금지 trim 후 저장
404 NOT_FOUND	정수 id stable error code

5xx · 서버 조사

error.code · 분기 기준

6.1. Endpoint inventory

method	path	목적	성공
`GET`	`/api/health`	local server 생존 확인	`200 {"ok":true}`
`GET`	`/api/observations`	최근 관찰 최대 20건	`200 data[] + meta.count`
`POST`	`/api/observations`	관찰 1건 생성	`201 data + Location`
`GET`	`/api/observations/{id}`	생성 결과 단건 확인	`200 data`

6.2. Stable error envelope

```
{
  "error": {
    "code": "VALIDATION_FAILED",
    "message": "입력값을 확인하세요.",
    "fields": {
      "summary": "앞뒤 공백을 제외하고 5자 이상 입력하세요."
    }
  }
}
```

속성	소비자 역할
`code`	프로그램이 분기하는 안정 식별자
`message`	사람이 보는 전체 설명
`fields`	입력 위치에 focus·오류 표시

프로그램이 한국어 message 문자열을 비교하게 하지 않습니다. 문구 수정과 번역이 client logic을 깨뜨릴 수 있기 때문입니다.

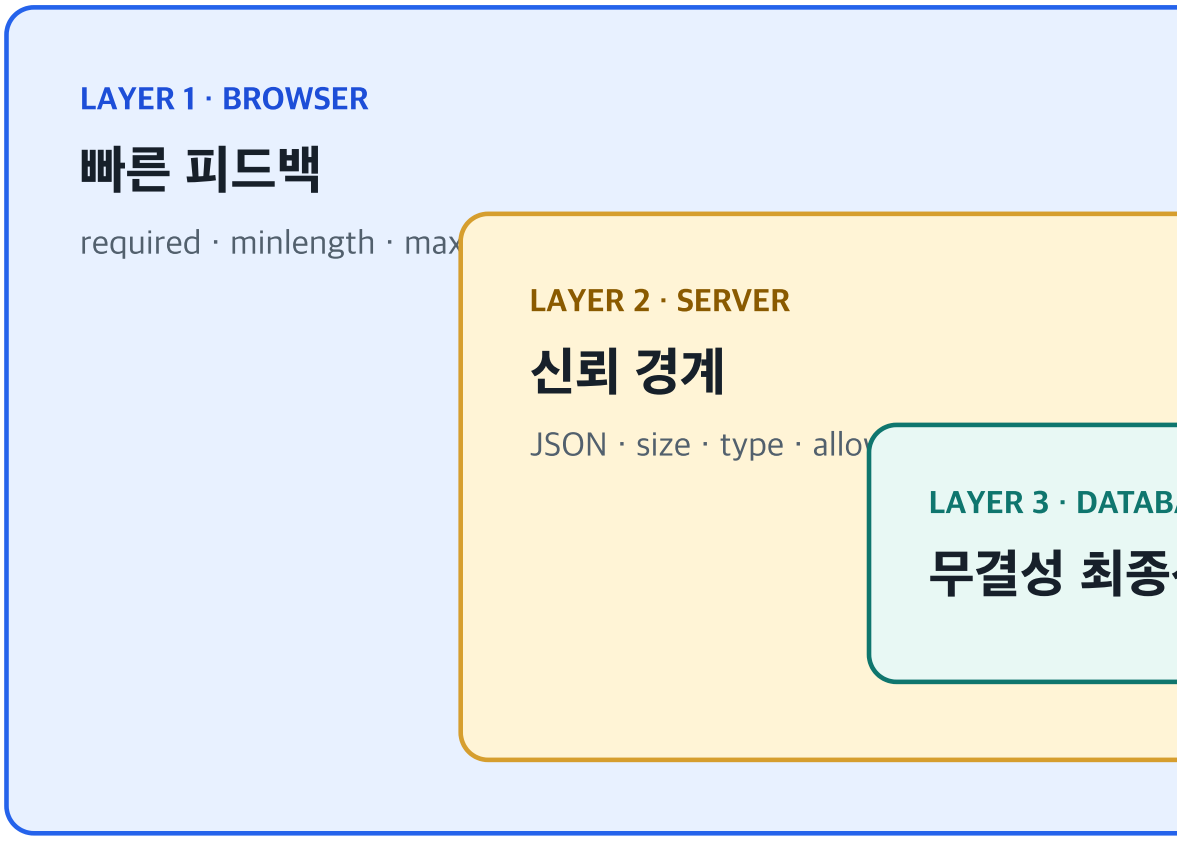
6.3. Status ownership

```
2xx: 요청과 결과가 계약대로 처리됨
4xx: client가 요청·입력·계약을 수정해야 함
5xx: server가 내부 실패를 조사해야 함
```

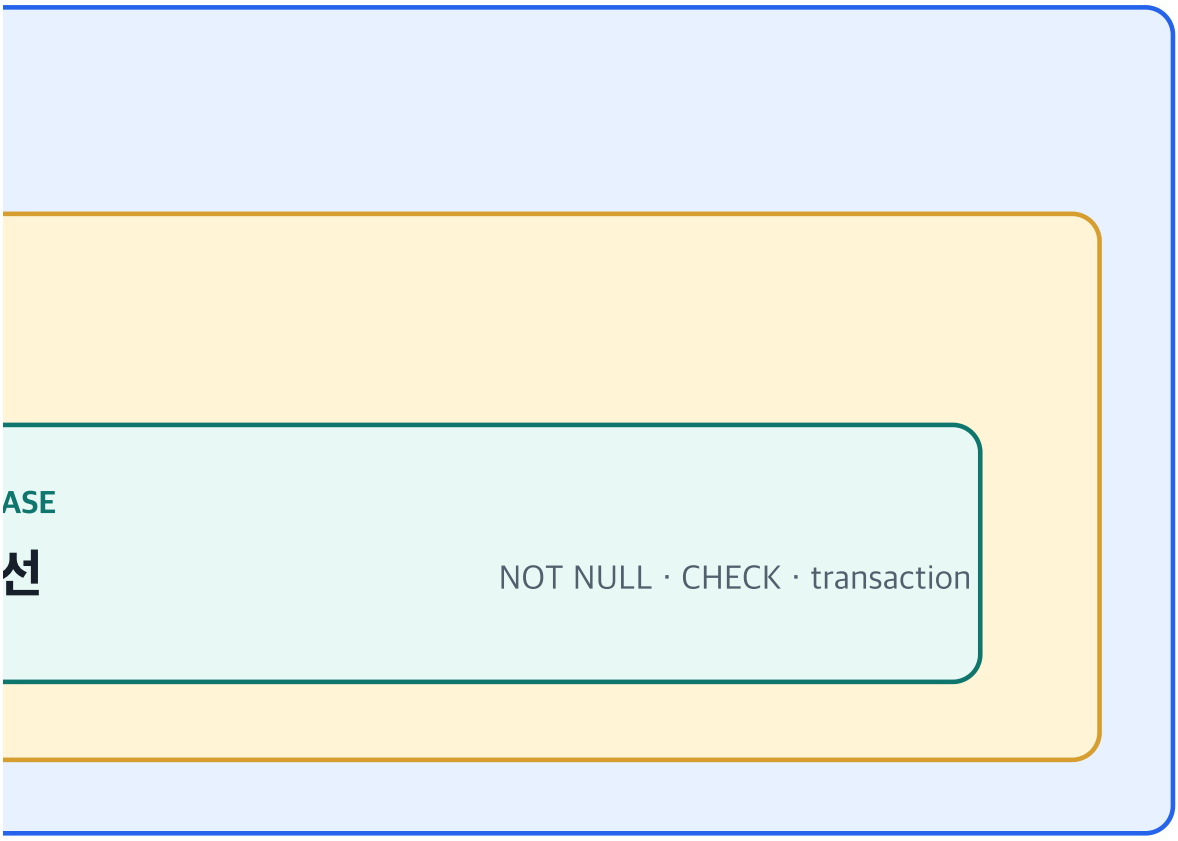
모든 실패를 `200 {ok:false}`로 돌려주면 `fetch`의 `response.ok`, monitoring, cache, proxy가 HTTP 의미를 사용할 수 없습니다.

7. 검증의 세 층을 역할로 분리합니다

그림 6. 같은 길이 규칙이 세 곳에 있어도 목적이 다릅니다. server validation이 보안과 업무 신뢰의 기준입니다.



! **브라우저 검증은 우회할 수 있으므로 서버 검증을 대신하지 못합니다**



ㅏ.

7.1. Browser validation

```
<textarea minlength="5" maxlength="80" required></textarea>
```

역할은 사용자가 왕복 전에 실수를 고치도록 돕는 것입니다. 개발자 도구, 직접 HTTP client, 변조된 JavaScript로 우회할 수 있습니다.

7.2. Server validation

```
if not isinstance(category, str) or category not in CATEGORIES:
    fields["category"] = "정해진 분류 코드 중 하나를 선택하세요."

normalized_summary = summary.strip()
if len(normalized_summary) < 5:
    fields["summary"] = "앞뒤 공백을 제외하고 5자 이상 입력하세요."
```

OWASP Input Validation Cheat Sheet는 외부에서 들어온 값을 가능한 이른 시점에 syntactic·semantic 수준으로 검증하고, 작은 고정 집합은 allowlist로 확인하도록 권고합니다. client-side 검증은 UX용이고 server-side 검증은 우회할 수 없는 기준이어야 합니다.

7.3. Database constraint

server bug, test fixture, maintenance script처럼 API를 거치지 않는 쓰기도 생길 수 있습니다. DB는 `NOT NULL`과 `CHECK`로 최소 무결성을 다시 요구합니다.

7.4. Normalize 뒤 validate

이번 규칙은 앞뒤 공백을 제거한 값을 기준으로 길이를 잽니다.

```
raw:        "  짧음  "  → 겉보기 6자
normalized: "짧음"      → 실제 2자
result: 422 VALIDATION_FAILED
```

원본을 보존해야 하는 domain이라면 normalize 정책이 달라집니다. 자동 trim이 항상 옳은 것은 아닙니다. 이번 관찰 요약 계약에서만 승인된 결정입니다.

8. Server 경계는 순서대로 좁혀 갑니다

Request handler는 “JSON을 받아 DB에 넣는 함수”가 아닙니다. 각 실패를 더 위험한 처리 전에 차단하는 순서가 있습니다.

```
1. path·method routing
2. Content-Type
3. Content-Length 존재·정수·4KB 이하
4. UTF-8 decode·JSON parse
5. object 여부·unknown field
6. type·allowlist·trim·length
7. repository 호출
8. DB integrity error mapping
9. generic internal error mapping
10. response serialization
```

8.1. 경계가 먼저인 이유

먼저 확인	뒤로 미루는 작업	피하는 위험
body size	body read·JSON parse	불필요한 memory·CPU 소비
JSON object	field access	예상하지 못한 list·scalar 처리
unknown field	model·DB 전달	mass assignment·contract drift
allowlist	업무 로직	조작된 option 값
validation	transaction	잘못된 data로 DB 점유

8.2. 오류 응답과 server log 분리

사용자 response에는 stack trace, filesystem path, SQL text를 넣지 않습니다.

```
response: INTERNAL_ERROR + 일반 문구
server log: exception trace + request method/path
금지: raw body·개인정보·secret 전체 logging
```

이번 실습은 합성 data만 쓰고 request body를 log하지 않습니다.

9. SQL 구조와 값을 Placeholder로 분리합니다

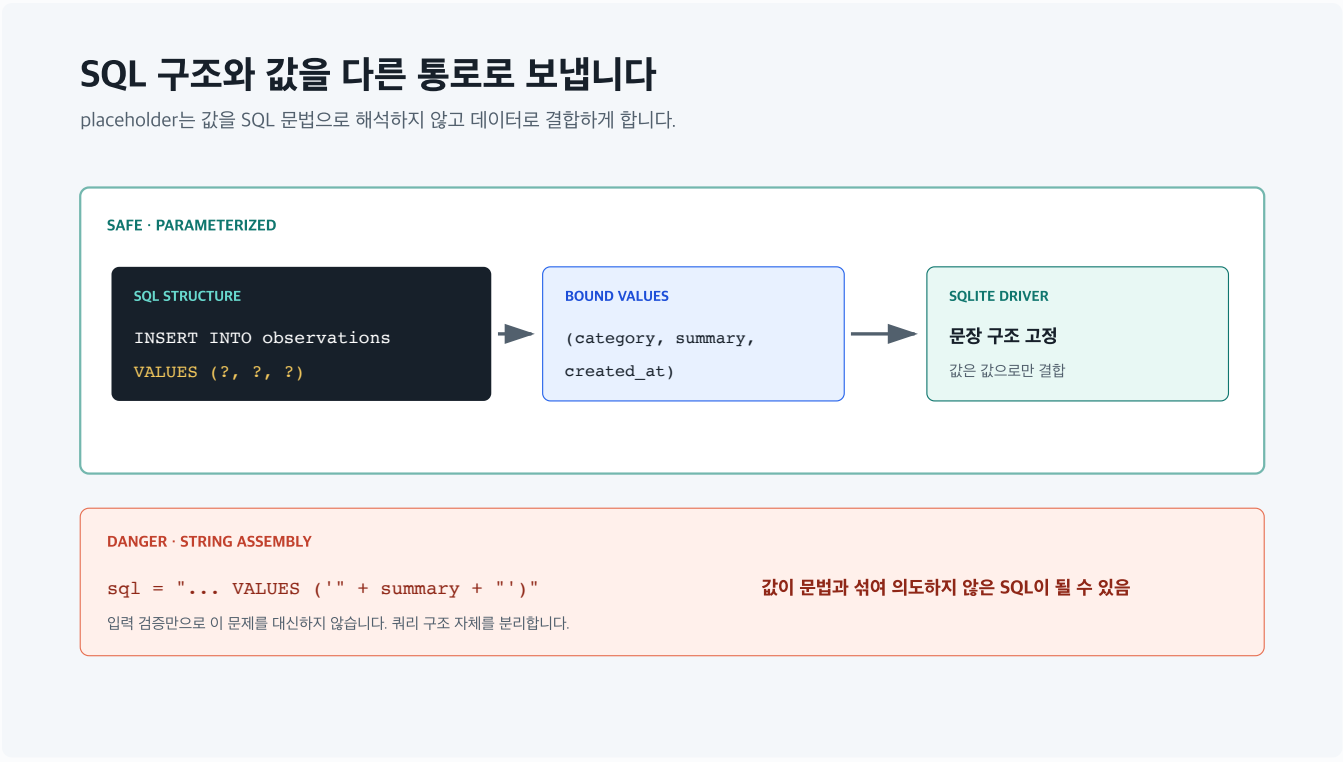


그림 7. 입력 검증과 parameterized query는 서로 대체하지 않습니다. 하나는 허용 의미를, 다른 하나는 SQL 구조와 값의 분리를 책임집니다.

SAFE · PARAMETERIZED

SQL STRUCTURE

```
INSERT INTO observations
VALUES (?, ?, ?)
```

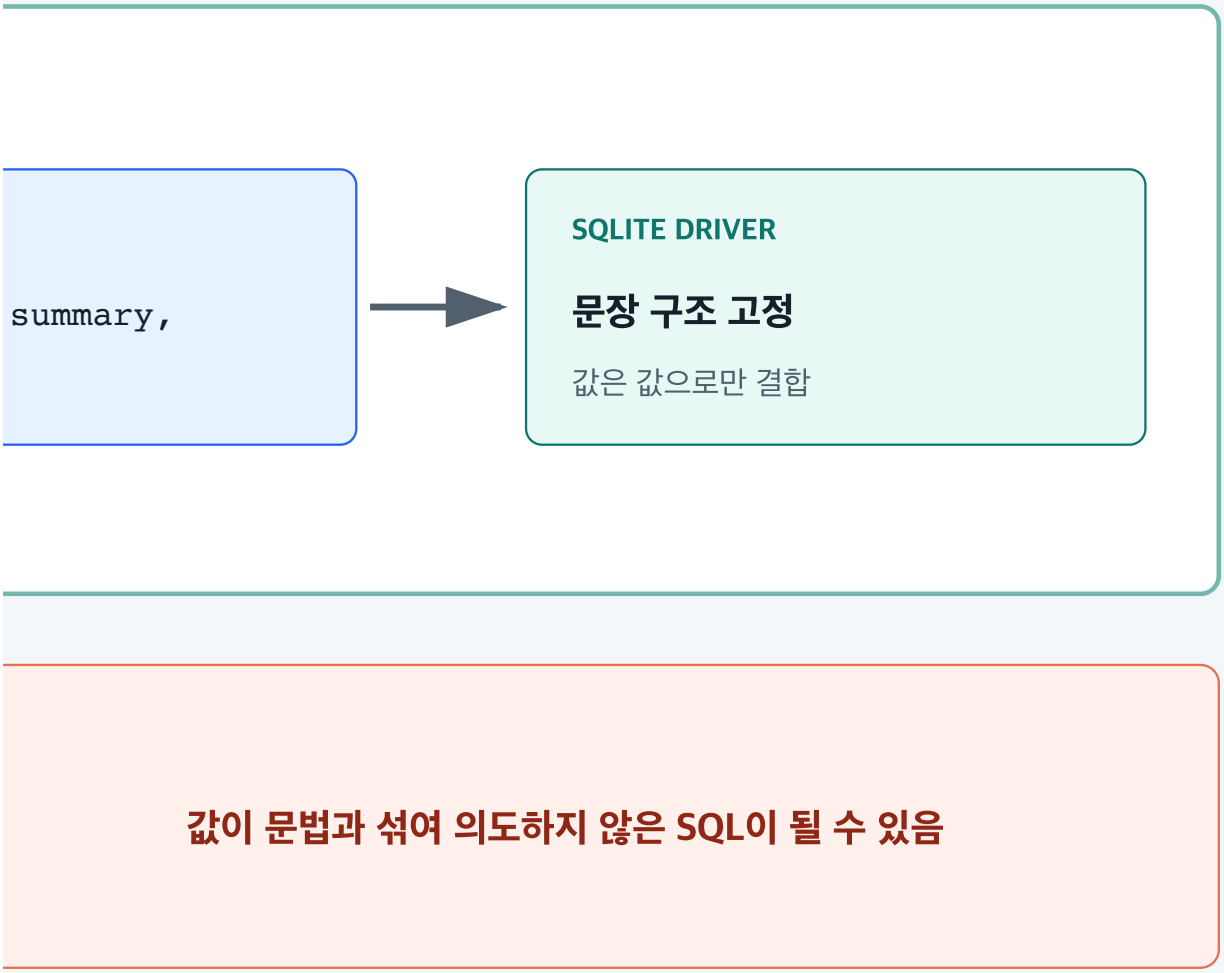
BOUND VALUES

```
(category,
created_at)
```

DANGER · STRING ASSEMBLY

```
sql = "... VALUES ('" + summary + "')"
```

입력 검증만으로 이 문제를 대신하지 않습니다. 쿼리 구조 자체를 분리합니다.



값이 문법과 섞여 의도하지 않은 SQL이 될 수 있음

9.1. 안전한 DB-API 호출

```
connection.execute(
    """
    INSERT INTO observations (category, summary, created_at)
    VALUES (?, ?, ?)
    """,
    (category, summary, created_at),
)
```

Python `sqlite3` 공식 문서는 SQL 문자열에 값을 직접 넣지 말고 placeholder로 parameter를 binding하도록 안내합니다. OWASP SQL Injection Prevention Cheat Sheet도 prepared statement와 parameterized query를 기본 방어로 둡니다.

9.2. 위험한 문자열 조립

```
# 하지 않습니다.
sql = "INSERT ... VALUES ('" + summary + "')"
```

입력값이 SQL 문법과 같은 문자열 안에 섞입니다. quote를 수동 escape하는 방식도 context와 driver 규칙을 놓치기 쉽습니다.

9.3. Placeholder가 해결하지 않는 것

- category가 업무 allowlist에 속하는지
- summary가 5~80자인지
- 현재 actor가 저장 권한이 있는지
- table·column 이름을 동적으로 허용해도 되는지
- 저장이 실제 outcome에 맞는지

값 binding은 SQL injection 위험을 줄이지만 업무 validation과 authorization을 대신하지 않습니다.

30초 확인 3
`summary`를 placeholder로 넣었으면 길이 검증을 생략해도 될까요? 두 방어가 답하는 질문을 각각 적습니다.

10. Transaction과 Connection 수명은 다릅니다

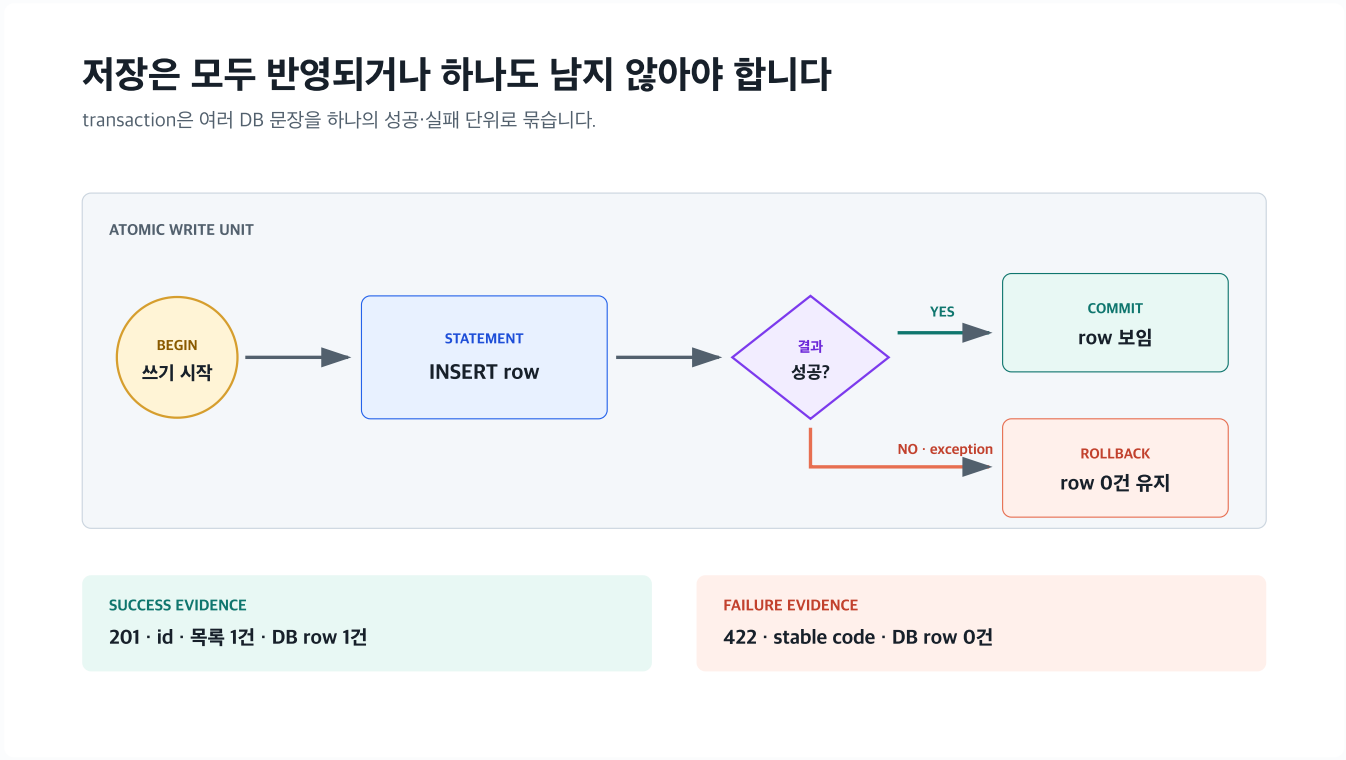
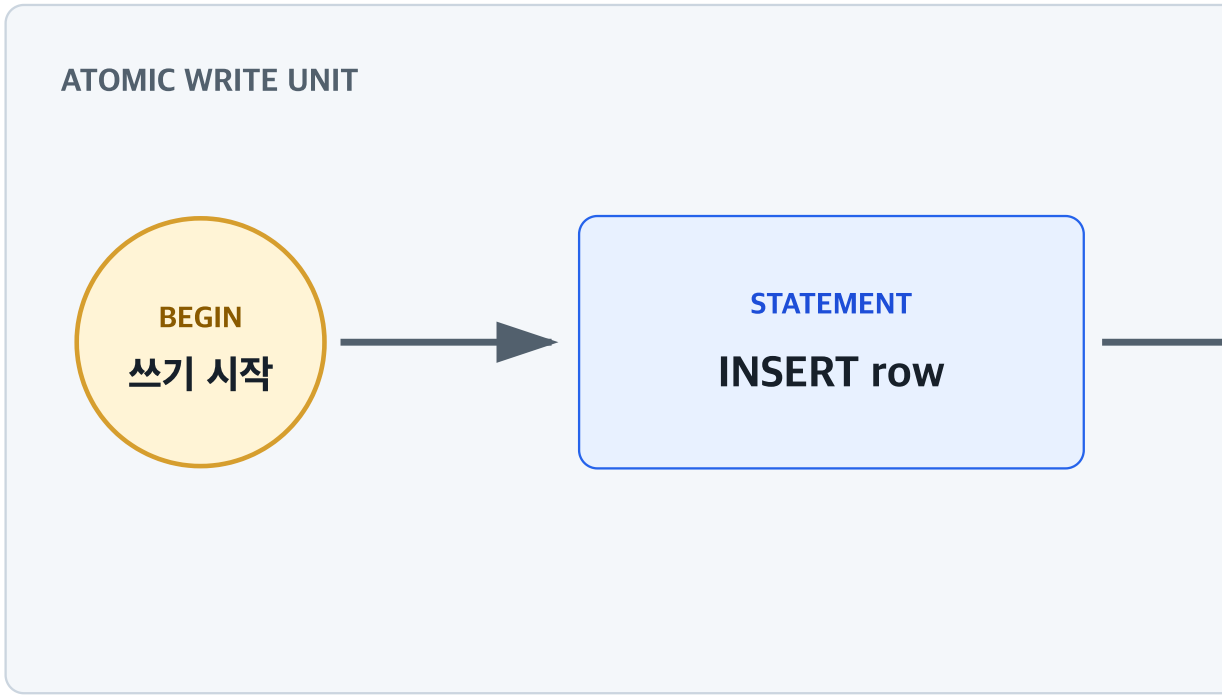
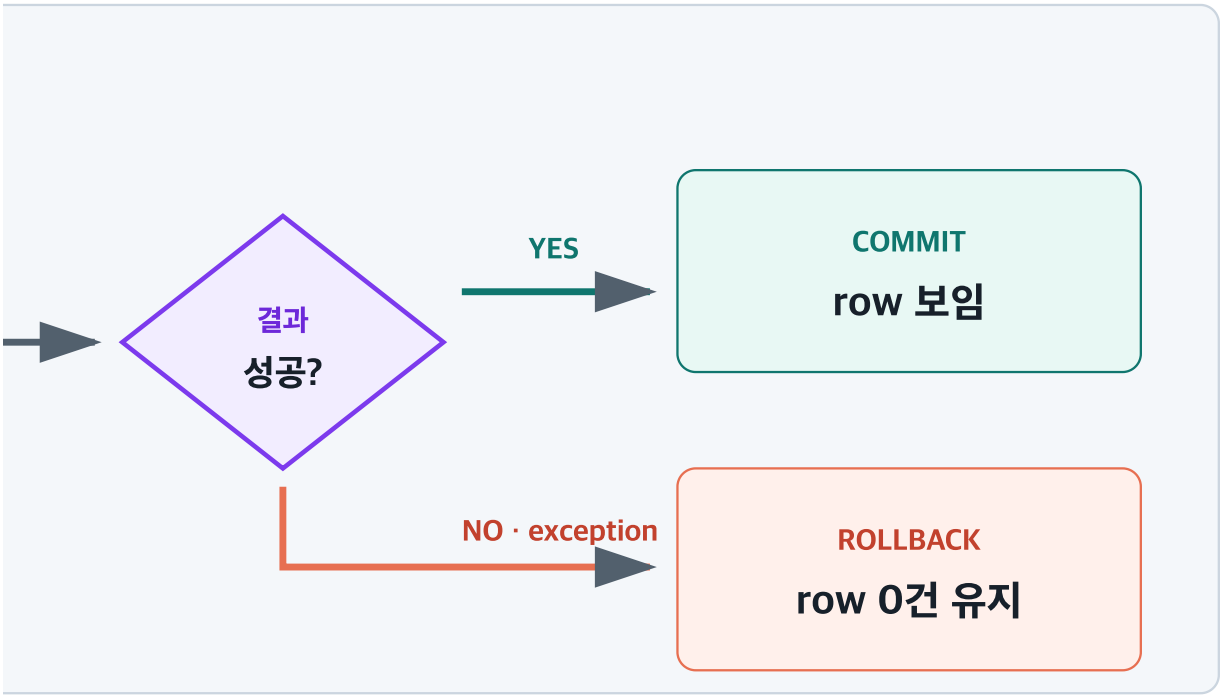


그림 8. transaction은 data 변화의 원자성을, connection closing은 resource 수명을 책임집니다.



SUCCESS EVIDENCE

201 · id · 목록 1건 · DB row 1건



FAILURE EVIDENCE

422 · stable code · DB row 0건

10.1. 두 개의 수명주기

```
from contextlib import closing

with closing(connect(db_path)) as connection:  # 끝에서 connection.close()
    with connection:                           # 성공 commit, exception rollback
        cursor = connection.execute(...)
        row = connection.execute(...).fetchone()
```

Python `sqlite3.Connection`의 context manager는 pending transaction을 commit 또는 rollback하지만 connection을 닫지 않습니다. 그래서 실습은 `closing()`과 `with connection`을 겹쳐 두 책임을 분명히 합니다.

10.2. 원자성 증거

장면	HTTP	DB row	판정
유효한 관찰	`201`	1건 증가	commit
summary 2자	`422`	변화 없음	transaction 시작 전 차단
DB constraint 실패	`422`	변화 없음	rollback
unexpected exception	`500`	partial row 없음	rollback + 조사

10.3. SQLite 동시성 범위

SQLite는 local·single-process 학습 MVP에 적합합니다. 한 시점에 writer는 하나이므로 `busy_timeout = 3000`을 두었지만, 이는 운영 부하 설계를 대신하지 않습니다. 높은 동시 쓰기, 여러 application instance, 복잡한 권한은 다음 architecture 결정입니다.

11. Schema를 마지막 무결성 계약으로 씁니다

그림 9. row가 실제로 존재한다는 사실은 최소한 모든 DB constraint를 통과했다는 증거입니다.

observations

COLUMN	TYPE	INTEGRITY RULE
id	INTEGER	PRIMARY KEY
category	TEXT	NOT NULL · typeof text IN (USABILITY, PERFORMAN
summary	TEXT	NOT NULL · trim 일치 length 5..80
created_at	TEXT	NOT NULL · ISO 문자열 최소

INDEX created_at DESC, id DESC · 최신 20건 조회

ICE, CONTENT)

길이

REJECTED AT DB

`category = "OTHER"`
CHECK allowlist 실패

`summary = "짧음"`
CHECK length 실패

`summary = " 값 "`
CHECK trim 실패

PASS CONDITION
row가 보이면 모든 제약 통과

```
CREATE TABLE IF NOT EXISTS observations (
  id INTEGER PRIMARY KEY,
  category TEXT NOT NULL
    CHECK (typeof(category) = 'text')
    CHECK (category IN ('USABILITY', 'PERFORMANCE', 'CONTENT')),
  summary TEXT NOT NULL
    CHECK (typeof(summary) = 'text')
    CHECK (summary = trim(summary))
    CHECK (length(summary) BETWEEN 5 AND 80),
  created_at TEXT NOT NULL
    CHECK (typeof(created_at) = 'text')
    CHECK (length(created_at) >= 20)
);
```

11.1. 각 constraint의 질문

constraint	막는 상태
`PRIMARY KEY`	row를 안정적으로 식별할 수 없음
`NOT NULL`	필수 값이 사라짐
`typeof(...)='text'`	SQLite 동적 type으로 예상 밖 값 저장
`IN (...)`	server allowlist 밖 category
`summary = trim(summary)`	저장 기준과 표시 기준 불일치
`length BETWEEN 5 AND 80`	너무 짧거나 긴 요약

11.2. Index는 query와 함께 설계합니다

```
CREATE INDEX idx_observations_newest
ON observations(created_at DESC, id DESC);
```

목록 query의 `ORDER BY created_at DESC, id DESC LIMIT ?`와 같은 정렬 방향입니다. 이번 작은 DB에서 성능 차이는 거의 없지만, query contract와 index 의도를 함께 읽는 습관을 배웁니다.

11.3. Schema가 부족한 부분

`created_at`을 TEXT로 저장하며 최소 길이만 확인하므로 완전한 ISO 8601 의미까지 DB가 검증하지는 않습니다. server가 UTC ISO string을 생성하고 test가 형식을 확인합니다. 학습 MVP의 의도적 단순화이며 template의

limit에 기록합니다.

12. Response를 Row와 화면 사이의 계약으로 만듭니다

생성 성공 response는 저장된 row를 그대로 증명하는 최소 표현입니다.

```
HTTP/1.1 201 Created
Content-Type: application/json; charset=utf-8
Location: /api/observations/3
Cache-Control: no-store
```

```
{
  "data": {
    "id": 3,
    "category": "USABILITY",
    "summary": "저장 완료 상태를 확인합니다.",
    "createdAt": "2026-07-16T02:09:10.123Z"
  }
}
```

12.1. 왜 input만 되돌리지 않는가

server와 DB가 만든 `id`, `createdAt`, trim된 `summary`를 client가 알아야 실제 저장 결과를 표시할 수 있습니다. client가 임의 id·시각을 만들면 화면과 DB가 갈라집니다.

12.2. Security headers

실습 server는 모든 response에 다음을 추가합니다.

header	학습 목적
`Content-Security-Policy`	같은 origin의 script·style·connect만 허용
`X-Content-Type-Options: nosniff`	선언 type을 임의 추측하지 않도록 함
`Referrer-Policy: no-referrer`	referrer 전달 억제
`Cross-Origin-Opener-Policy: same-origin`	browsing context 분리

이 headers가 production hardening 전체를 의미하지는 않습니다. TLS, authentication, CSRF, rate limit, secure cookies는 현재 non-goal입니다.

13. Fetch는 Network 성공과 HTTP 성공을 구분합니다

MDN Fetch 문서의 핵심 함정은 `404`나 `422`도 `fetch()` Promise 자체는 resolve될 수 있다는 점입니다. 반드시 `response.ok`를 확인합니다.

```
async function fetchJson(url, options = {}) {
  const response = await fetch(url, options);
  const payload = await response.json().catch(() => ({}));

  if (!response.ok) {
    const error = new Error(payload.error?.message || `HTTP ${response.status}`);
    error.code = payload.error?.code || "HTTP_ERROR";
    error.fields = payload.error?.fields || {};
    throw error;
  }
  return payload;
}
```

13.1. 두 종류의 실패

실패	예	확인
network·browser 실패	server 꺼짐, 연결 거부	`fetch` 자체 reject
HTTP application 실패	`400`, `415`, `422`, `500`	response resolve 후 `ok=false`

13.2. JSON parse 실패 처리

`response.json()`도 실패할 수 있습니다. 실습은 body가 비거나 JSON이 아닐 때 빈 object로 대체한 뒤 status 기반 일반 오류를 만듭니다. production에서는 content type과 error observability를 더 엄격히 설계합니다.

13.3. Same-origin으로 시작합니다

UI와 API를 같은 `127.0.0.1:4173`에서 제공해 첫 slice에서 CORS를 제외합니다. CORS 설정을 배우기 전에 불필요한 origin 분리를 만들지 않습니다.

14. 화면을 네 상태와 생성 Subflow로 설계합니다

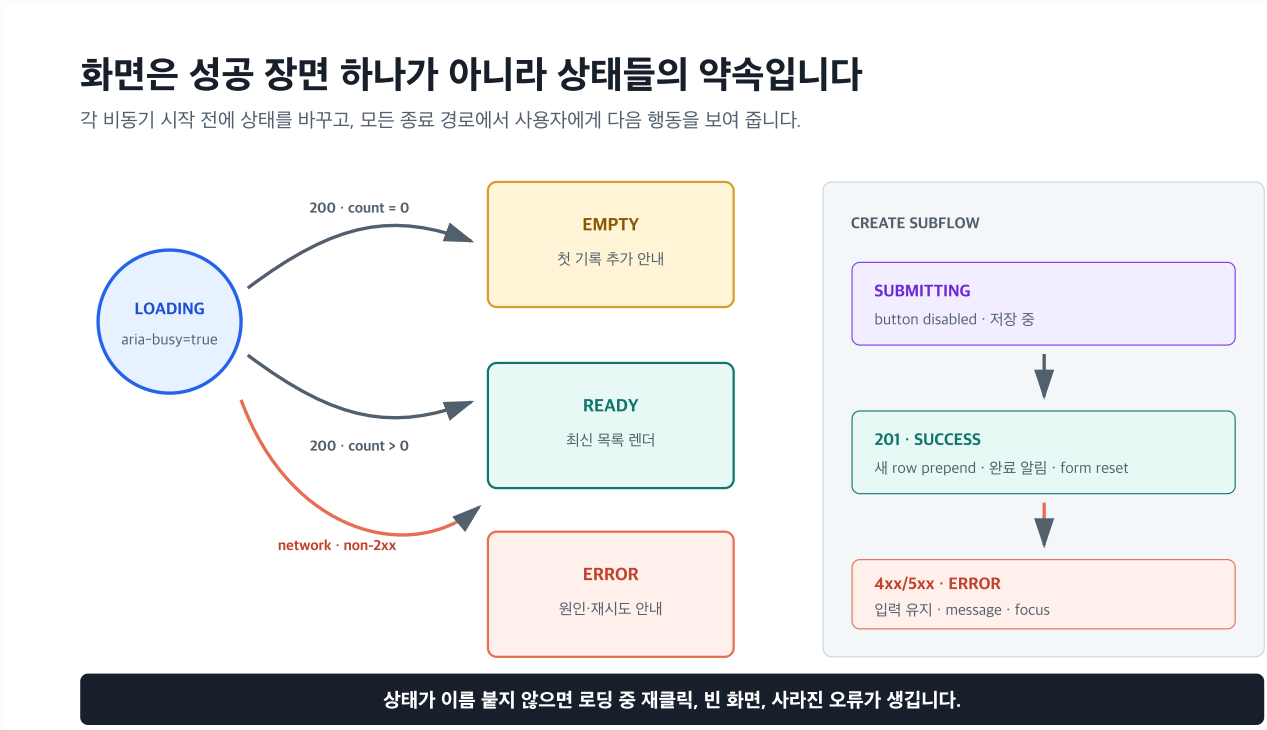
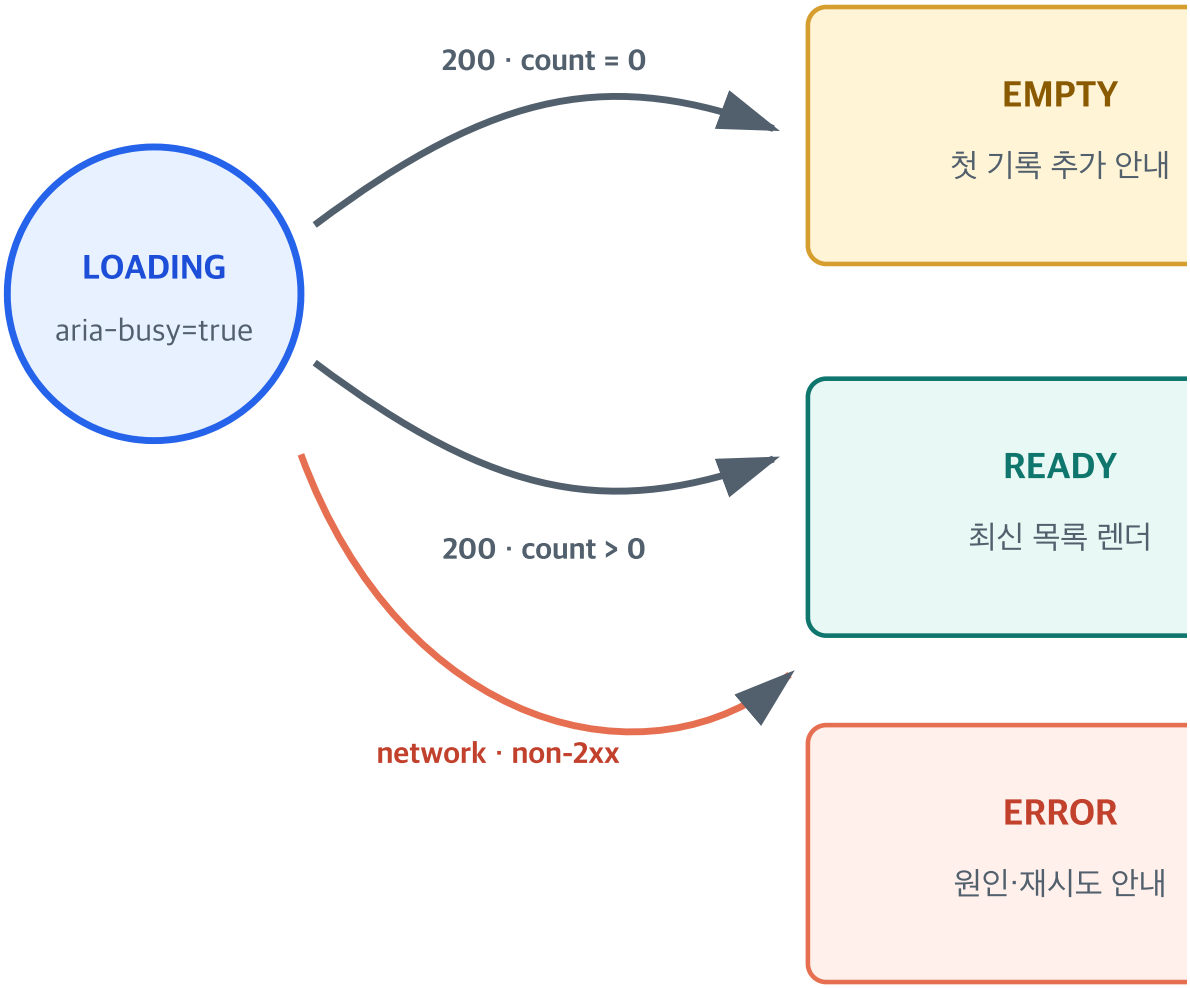
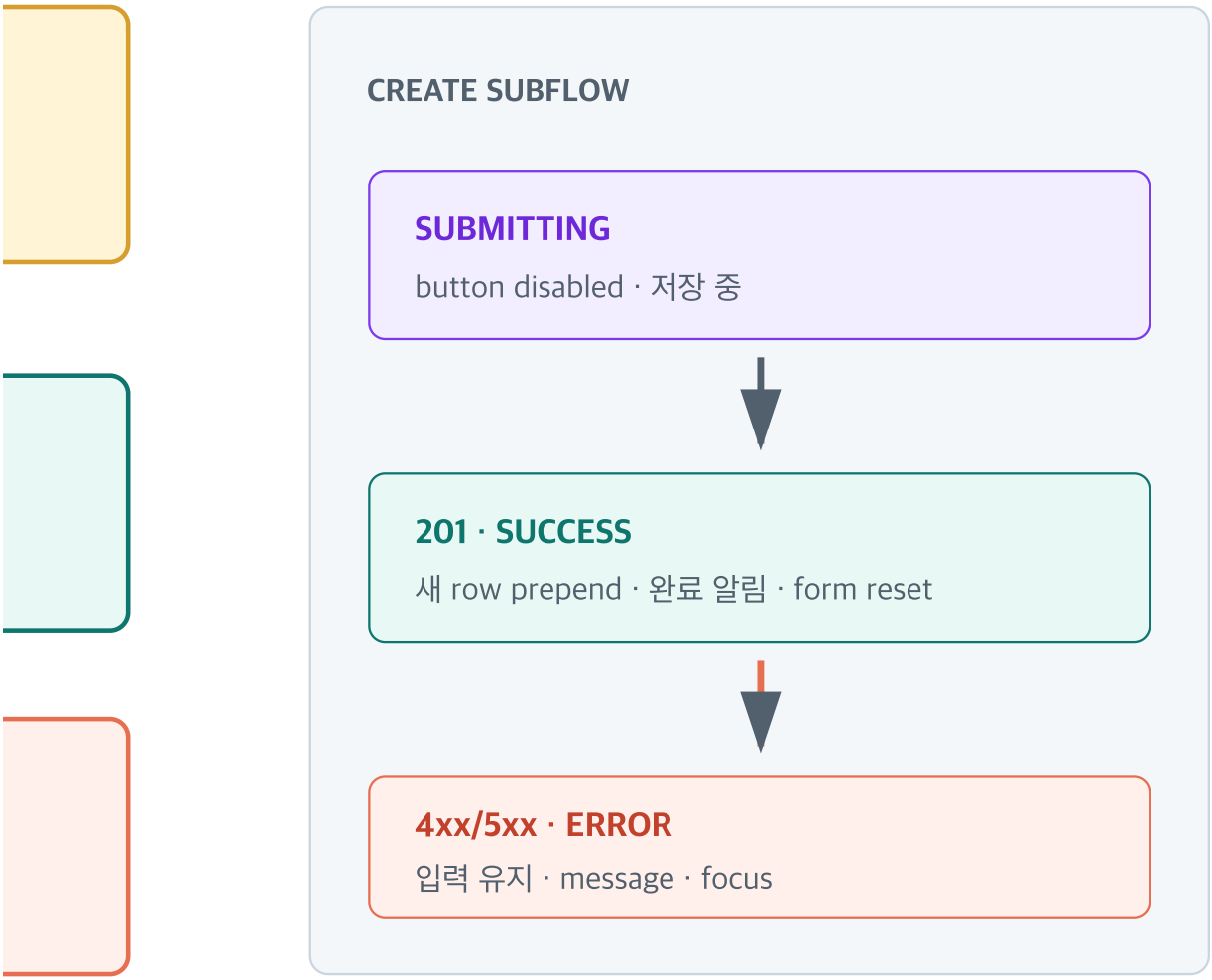


그림 10. 상태를 이름 붙이면 중복 제출, 영원한 loading, 이유 없는 빈 화면, 사라진 오류를 test할 수 있습니다.


LOADING
aria-busy=true
200 · count = 0
200 · count > 0
network · non-2xx
EMPTY
첫 기록 추가 안내
READY
최신 목록 렌더
ERROR
원인·재시도 안내


CREATE SUBFLOW
SUBMITTING
button disabled · 저장 중
201 · SUCCESS
새 row prepend · 완료 알림 · form reset
4xx/5xx · ERROR
입력 유지 · message · focus

14.1. 목록 상태

상태	진입 조건	화면	나가는 조건
`loading`	최초 조회 시작	로딩 문구, `aria-busy=true`	200 또는 오류
`empty`	200 + 0건	첫 기록 추가 안내	생성 성공·재조회
`ready`	200 + 1건 이상	최근 목록	생성·재조회·오류
`error`	network 또는 non-2xx	원인 문구	재시도·server 복구

실제 code는 `empty`를 별도 string으로 저장하지 않고 `ready + length===0`에서 렌더합니다. 개념 상태와 구현 표현이 반드시 1:1일 필요는 없지만 판정 조건은 명확해야 합니다.

14.2. 생성 상태

```
idle → submitting → success → idle
                  ↘ error   → idle
```

`submitting` 동안 button을 disabled로 해 accidental double click을 줄입니다. 이것만으로 idempotency를 보장하지는 않습니다. network retry와 중복 요청을 운영에서 다루려면 idempotency key나 unique business key가 필요합니다.

14.3. 실패해도 입력을 보존합니다

성공하면 form을 reset하지만 실패하면 사용자가 입력한 값을 유지합니다. `error.fields`의 첫 field로 focus를 이동해 수정 위치를 알려 줍니다.

15. 응답 값을 Text로 렌더합니다

서버가 검증한 문자열도 HTML로 실행할 이유는 없습니다. 관찰 요약은 text로 표시합니다.

```
fragment.querySelector(".summary").textContent = observation.summary;
```

15.1. `innerHTML`을 쓰지 않는 이유

```
// 이번 실습에서 금지
element.innerHTML = observation.summary;
```

사용자 제어 값이 HTML parser로 들어가면 markup과 script context 위험이 생깁니다. `textContent`는 문자열을 text node로 취급합니다. OWASP도 user-controlled data를 출력 context에 맞게 encoding하도록 안내합니다.

15.2. Safe rendering audit

실습 감사 script는 다음을 자동 확인합니다.

```
textContent present
innerHTML absent
response.ok check present
Content-Security-Policy present
runtime URL은 loopback-only
```

15.3. 접근 가능한 상태 알림

```
<div id="notice" role="status" aria-live="polite"></div>
```

화면 색만 바꾸지 않고 text message와 live region을 함께 사용합니다. loading 목록에는 `aria-busy`를 둡니다.

16. 오류를 경계별 소유권으로 좁힙니다

오류는 마지막으로 통과한 경계에서 좁혀 갑니다

"안 돼요"를 재현 조건·경계·증거·다음 행동으로 바꾸면 조사 범위가 작아집니다.

경계	대표 신호	확인 증거	다음 행동
브라우저	required · offline · JS exception	console · request 시작 여부	입력 수정 · 연결 확인
HTTP	404 · 405 · 415 · 413	method · URL · headers · size	계약에 맞게 요청 수정
서버 검증	422 + VALIDATION_FAILED	error.code · error.fields	사용자 값·UI 규칙 수정
데이터베이스	constraint · lock · transaction	row count · schema · server log	원자성 확인 · 원인 제거
응답·렌더	2xx인데 목록 미갱신	response body · client state · DOM	data→view mapping 수정

status만 보지 말고 stable code와 DB 변화 여부를 함께 기록합니다.

그림 11. status와 message만 보지 말고 request가 어디까지 통과했는지, DB row가 바뀌었는지를 함께 봅니다.

경계	대표 신호	확인 증거
브라우저	required · offline · JS exception	console
HTTP	404 · 405 · 415 · 413	method
서버 검증	422 + VALIDATION_FAILED	error.co
데이터베이스	constraint · lock · transaction	row cou
응답·렌더	2xx인데 목록 미갱신	respons

	다음 행동
· request 시작 여부	입력 수정 · 연결 확인
· URL · headers · size	계약에 맞게 요청 수정
de · error.fields	사용자 값·UI 규칙 수정
ınt · schema · server log	원자성 확인 · 원인 제거
e body · client state · DOM	data→view mapping 수정

16.1. 최소 진단 문장

```
Given: reset 후 합성 seed 2건, server 127.0.0.1:4173
When: category=OTHER로 POST /api/observations
Then: 422 VALIDATION_FAILED, category field error, DB count 변화 없음
Last passed boundary: JSON parse
Failed boundary: server semantic validation
Next action: client option과 API allowlist 대조
```

16.2. “저장 안 됨”을 좁히는 순서

1. 브라우저 request가 시작됐는가
2. method·URL·Content-Type이 계약과 같은가
3. status와 `error.code`는 무엇인가
4. response body에 생성 `id`가 있는가
5. 단건 GET에서 같은 `id`가 보이는가
6. 목록 GET count가 늘었는가
7. client state에 row가 들어왔는가
8. DOM에 text로 렌더됐는가

16.3. Generic 500의 한계

사용자에게 상세 내부 오류를 숨기는 것과 운영자가 원인을 모르는 것은 다릅니다. 이번 local MVP는 console log만 사용합니다. 운영에서는 correlation ID, structured log, metric, trace, alert owner가 필요합니다.

17. 테스트를 계약 추적표로 구성합니다

테스트 수보다 계약을 가로지르는 추적이 중요합니다

각 행은 하나의 약속이고, 각 열은 그 약속을 확인하는 서로 다른 관찰점입니다.

TRACE MATRIX

계약	VALIDATION	DB	API	FRONTEND	BROWSER
유효한 관찰 1건 저장	●	●	●	○	●
알 수 없는 category 거절	●	●	●	○	○
실패하면 row 0건	○	●	●	○	○
loading·empty·error 표시	○	○	○	●	●
저장 뒤 목록 첫 줄 표시	○	●	●	●	●

● 직접 증거 ○ 다른 층의 간접 증거

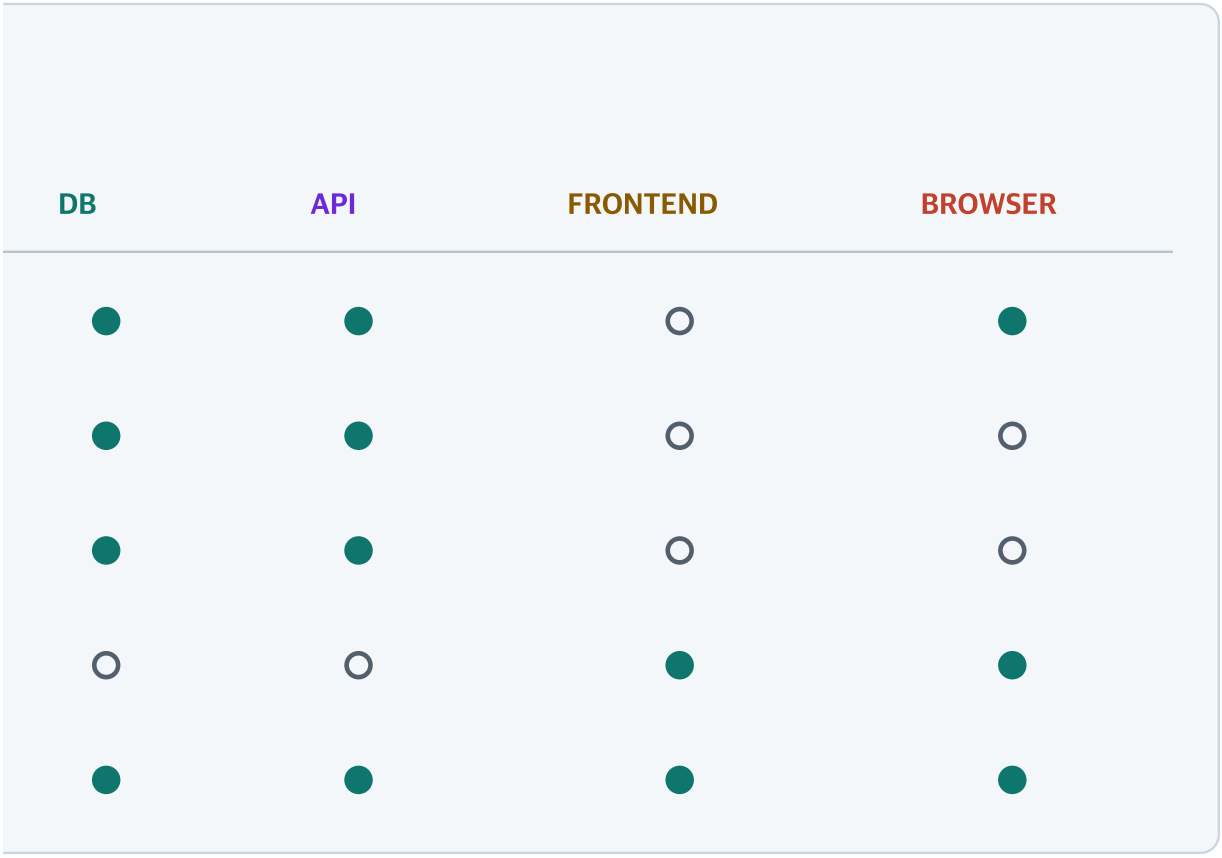
실습: 18 tests + 13 audit + HTTP probe

그림 12. 한 종류의 test만 많아서는 왕복을 증명하지 못합니다. 같은 계약을 서로 다른 관찰점에서 확인합니다.

TRACE MATRIX

계약	VALIDATION
유효한 관찰 1건 저장	●
알 수 없는 category 거절	●
실패하면 row 0건	○
loading·empty·error 표시	○
저장 뒤 목록 첫 줄 표시	○

● 직접 증거　　○ 다른 층의 간접 증거



실습: 18 tests + 13 audit + HTTP probe

17.1. 실습의 자동 증거

묶음	test	확인
validation unit	4	trim, allowlist, short text, unknown field
database	4	create/list, constraint, rollback, 20 limit
API integration	7	health, 201, 400, 413, 415, 422, 404·405
frontend contract	3	four states, safe DOM, native constraints
합계	**18**	Python 3.12·3.14 모두 PASS

17.2. 구조 감사 13개

```
required files
safe DOM
fetch response.ok
parameterized INSERT
transaction context
connection close
server validation
4KB body limit
security headers
DB constraints
local default
no external runtime URL
synthetic boundary
```

17.3. HTTP contract probe

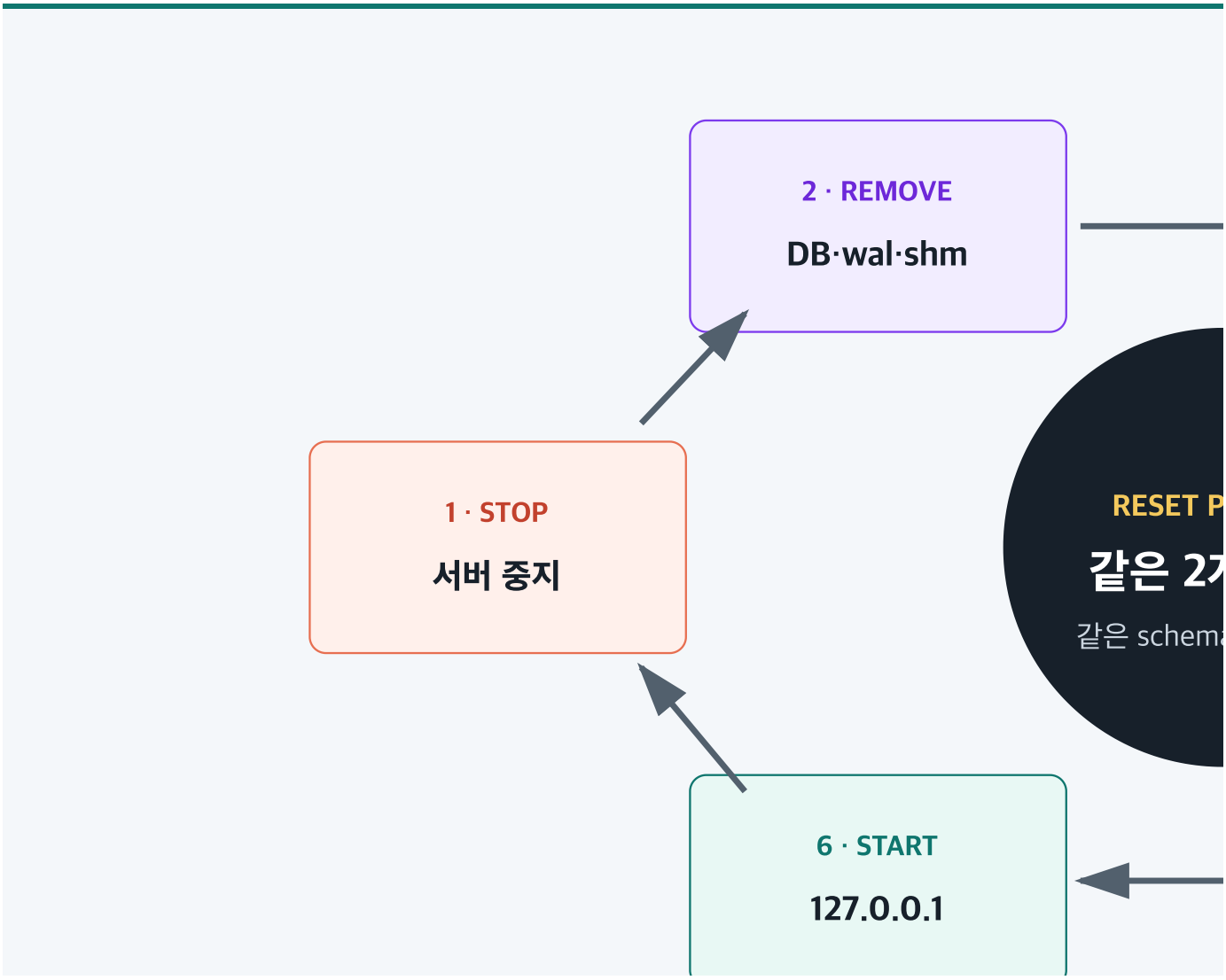
실제 ephemeral port server를 시작하고 다음을 연속 실행합니다.

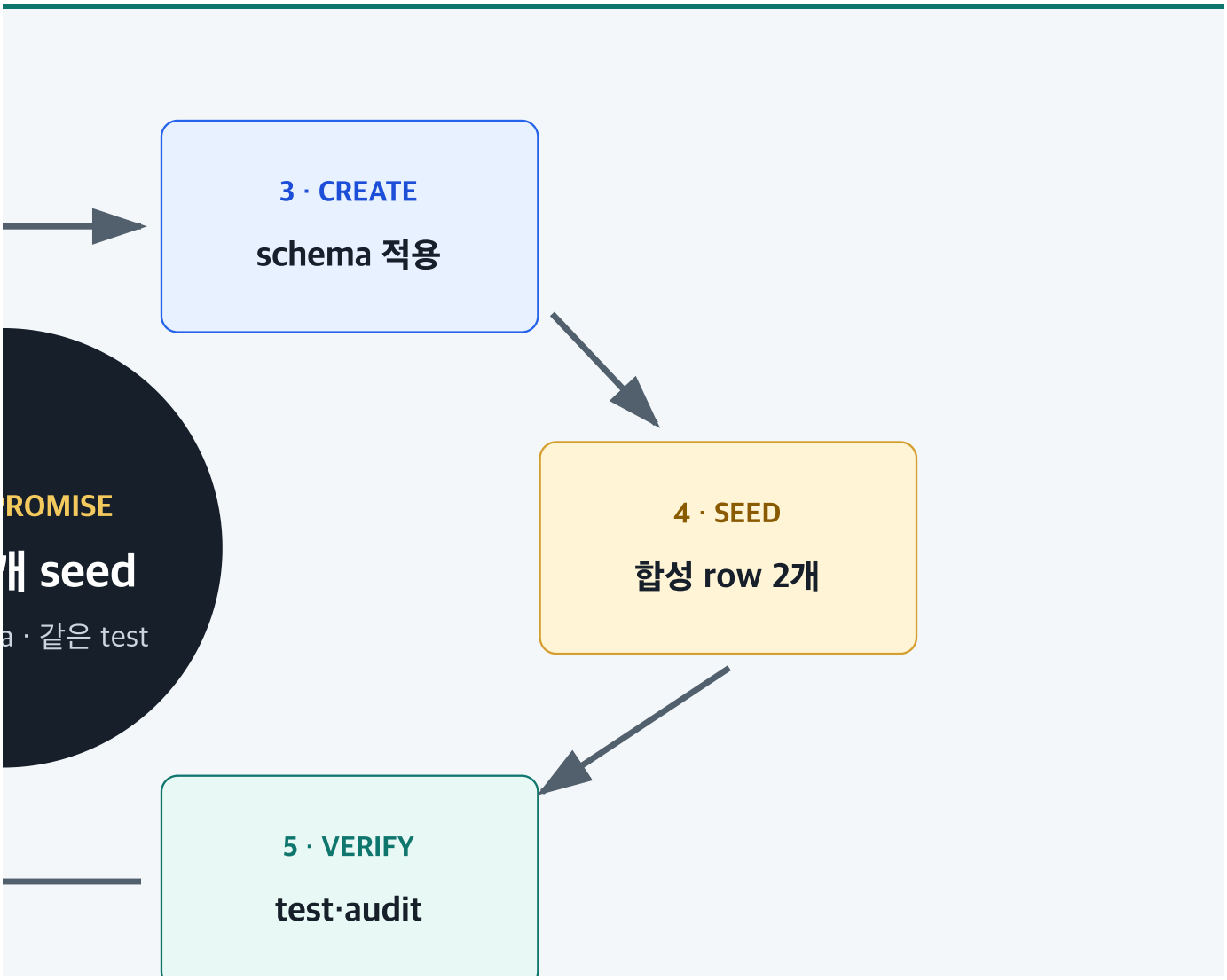
```
health 200
empty list count 0
invalid category 422
valid create 201
list count 1
item GET 200
result PASS
```

unit test가 함수 규칙을, API integration이 HTTP 경계를, probe가 사용자 왕복에 가까운 계약을 확인합니다.

18. Reset을 재현 가능한 복구 절차로 만듭니다

그림 13. reset 성공은 파일이 사라졌다는 뜻이 아니라 같은 schema와 seed, 검증 결과로 돌아왔다는 뜻입니다.


2 · REMOVE
DB·wal·shm
1 · STOP
서버 중지
RESET P
같은 2
같은 schem
6 · START
127.0.0.1


3 · CREATE
schema 적용
4 · SEED
합성 row 2개
5 · VERIFY
test·audit
ROMISE
개 seed
a · 같은 test

18.1. Exact reset

```
cd gibalja-fullstack-mvp-practice/app
python3 scripts/reset_db.py
```

기대 출력:

```
RESET PASS: .../var/observations.db
SYNTHETIC ROWS: 2
```

18.2. Reset이 하는 일

```
1. observations.db 삭제
2. observations.db-wal 삭제
3. observations.db-shm 삭제
4. schema.sql 적용
5. 고정 시각의 합성 row 2개 삽입
6. 목록 count 2 확인
```

server가 파일을 사용 중일 수 있으므로 먼저 중지합니다. `-wal`, `-shm` sidecar까지 같은 DB 상태에 속합니다.

18.3. Reset과 migration을 구분합니다

reset	migration
local 합성 data를 버리고 재생성	보존해야 할 data를 새 schema로 이동
학습·test용	운영·공유 environment용
deterministic seed	forward·rollback·backup 계획
실제 data에 적용 금지	owner·change window 필요

19. 완성 화면에서 Evidence를 읽습니다

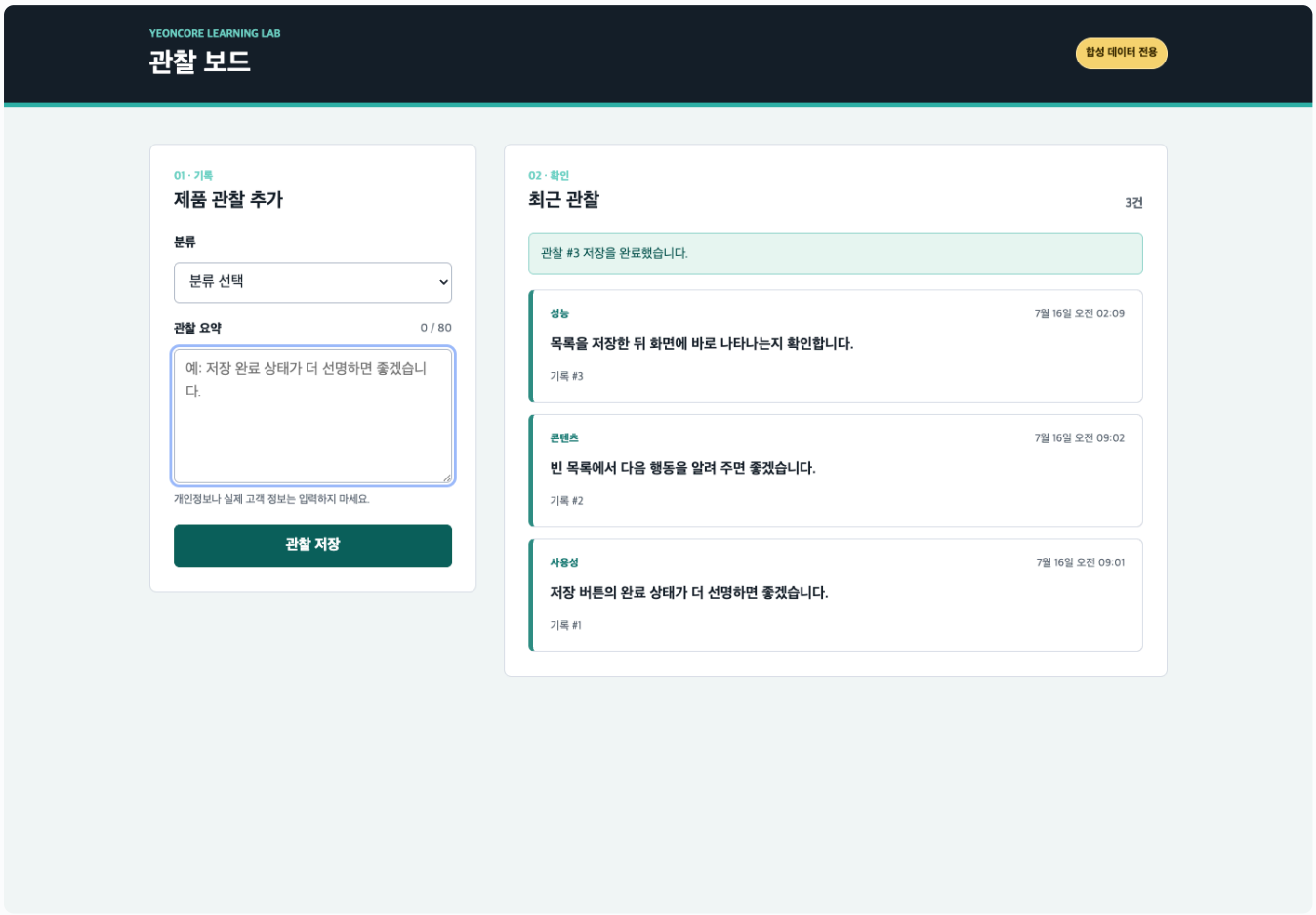


그림 15. Playwright로 실제 `POST`를 실행한 직후 화면입니다. 201 응답, `관찰 #3`, 3건 count, 완료 알림이 같은 저장을 가리킵니다.

19.1. 화면에서 보이는 계약

영역	보이는 값	뒤의 증거
합성 data badge	실제 정보 입력 금지	AGENTS.md boundary
분류 select	3개 label	stable code allowlist
0 / 80	입력 길이	browser + server + DB rule
저장 완료 알림	생성 `id=3`	response.data.id
최근 관찰 3건	seed 2 + 생성 1	list meta.count
목록 첫 항목	trim된 요약	DB row→JSON→textContent

19.2. Browser validation 결과

viewport	POST	horizontal overflow	console error	request failure
desktop 1440×1000	201	0	0	0
mobile 390×844	201	0	0	0

반응형에서 화면이 한 열로 바뀌어도 form과 list가 겹치지 않고, 같은 기능을 수행합니다.

19.3. 이 화면이 증명하지 않는 것

- 다른 사용자의 동시 입력
- authentication·authorization
- production traffic과 장애 복구
- backup·retention·privacy compliance
- 인터넷 공개 보안

좋은 evidence packet은 PASS뿐 아니라 증명 범위의 한계도 함께 적습니다.

20. 실습 Evidence Packet을 읽습니다

생성 완료 후 `evidence/`에는 여섯 파일이 있습니다.

파일	claim
`01-reset.txt`	같은 합성 시작점 2건
`02-tests.txt`	18개 behavior·boundary test PASS
`03-audit.txt`	13개 source·safety gate PASS
`04-contract-probe.json`	실제 HTTP 왕복 PASS
`05-database-snapshot.json`	schema와 현재 row
`06-versions.txt`	Python·SQLite 실행 version

20.1. Evidence의 세 속성

```
traceable: 어떤 claim과 연결되는가
reproducible: exact command와 working directory가 있는가
bounded: 무엇을 확인하지 않았는가
```

20.2. 실행 순서

```
python3 scripts/reset_db.py
python3 -m unittest discover -s tests -v
python3 scripts/audit_mvp.py
python3 scripts/contract_probe.py
python3 app.py --host 127.0.0.1 --port 4173
```

각 command의 성공을 다음 command가 덮어쓰지 않게 개별 exit code와 output을 남깁니다.

20.3. Version evidence

이 교재는 다음 두 local 조합에서 생성기 전체를 검증했습니다.

```
Python 3.12.13 · SQLite 3.50.4
Python 3.14.5  · SQLite 3.53.2
```

Python 3.12+가 아니라 생성기는 `3.11+`를 요구합니다. 사용한 표준 기능이 3.11에서 제공되기 때문입니다. 다만 이 검수에서 직접 실행한 version과 지원 선언을 구분합니다.

21. 60분 실습 Mission

단계별 실습서에 더 자세한 기록 칸이 있습니다.

Mission 1 · 생성과 증거 확인 · 10분

```
./02_Labs/G08_Fullstack/L08-01_create-fullstack-mvp-practice.sh
cd gibalja-fullstack-mvp-practice/app
```

- [ ] `evidence/02-tests.txt`의 `Ran 18 tests`와 `OK`
- [ ] `evidence/03-audit.txt`의 `RESULT: 13/13 PASS`
- [ ] `evidence/04-contract-probe.json`의 `"result": "PASS"`

Mission 2 · 실제 저장 · 10분

```
python3 app.py --host 127.0.0.1 --port 4173
```

브라우저에서 합성 관찰을 저장하고 다음을 기록합니다.

```
request status =
response id =
Location =
목록 count 전 / 후 =
완료 notice =
```

Mission 3 · 세 실패 만들기 · 15분

브라우저 UI를 바꾸지 말고 test·probe에서 다음을 확인합니다.

input	expected	DB 변화
malformed JSON	`400 INVALID_JSON`	0
`text/plain`	`415 UNSUPPORTED_MEDIA_TYPE`	0
`category=OTHER`	`422 VALIDATION_FAILED`	0

Mission 4 · 코드에서 경계 찾기 · 15분

다음 위치를 연결 지도에 표시합니다.

```
public/index.html       field·native constraints
public/app.js           fetch·state·textContent
mvp/server.py           HTTP boundary·error mapping
mvp/validation.py       allowlist·normalize·length
mvp/db.py               placeholder·transaction·serialization
schema.sql              final integrity
```

Mission 5 · Reset과 재현 · 10분

server를 중지하고 reset한 뒤 목록이 다시 seed 2건인지 확인합니다. 다른 폴더 이름으로 생성기를 다시 실행해 동일한 evidence가 나오는지 확인합니다.

22. 자주 실패하는 연결 패턴

22.1. 성공 화면부터 만들기

신호: API가 꺼져도 sample card가 보입니다.
원인: client mock과 실제 response 경계가 섞였습니다.
수정: initial list를 빈 array로 두고 GET response만 state에 넣습니다.

22.2. Client validation만 믿기

신호: UI에서는 막히지만 직접 POST하면 잘못된 row가 생깁니다.
수정: server에서 type·allowlist·length를 다시 검증하고 negative API test를 둡니다.

22.3. 모든 오류를 500으로 처리하기

신호: 사용자가 고칠 입력과 server 장애를 구분하지 못합니다.
수정: parse `400`, media `415`, semantic `422`, unexpected `500`으로 ownership을 나눕니다.

22.4. SQL 문자열에 값을 붙이기

신호: quote가 들어간 요약에서 query가 깨지거나 injection 위험이 생깁니다.
수정: query structure와 bound tuple을 분리합니다.

22.5. Transaction과 close를 하나로 생각하기

신호: commit은 됐지만 connection resource가 늦게 정리됩니다.
수정: `closing(connection)`과 `with connection`을 각각 사용합니다.

22.6. Loading 상태에서 중복 submit

신호: 같은 관찰 row가 두 번 생깁니다.
수정: submit 중 button을 disable합니다. 운영 retry까지 다루려면 별도 idempotency 설계를 추가합니다.

22.7. 2xx인데 화면이 안 바뀜

신호: DB와 response에는 row가 있지만 목록이 이전 상태입니다.
수정: `state.observations = [payload.data, ...state.observations]`와 `render()`를 확인합니다.

22.8. Reset이 현재 위치에 따라 실패

신호: 다른 working directory에서 DB path가 달라집니다.
수정: script file 기준 absolute `ROOT`와 `DEFAULT_DB_PATH`를 계산합니다.

23. 일곱 Done Gate로 완료를 판정합니다

그림 14. 7/7 PASS, non-goal 유지, 다른 학습자의 처음부터 재현이 모두 충족될 때 완료입니다.

GATE 1 ✓

결과

저장한 값이 바로 보임

GATE 2 ✓

계약

UI·API·DB 규칙 정렬

GATE 5 ✓

안전

합성 data·boundaries

GATE 6

재현

start·test·reset

DONE

7/7 PASS + non-goal 유지

GATE 3 ✓

데이터

transaction·constraint

GATE 4 ✓

오류

4xx·5xx·화면 상태

✓

정확

GATE 7 ✓

증거

test·probe·screenshot

RULE

+ 다음 사람이 처음부터 재현

gate	PASS 질문	이번 evidence
RESULT	저장한 값이 곧바로 보이는가	browser POST 201·목록 첫 줄
CONTRACT	UI·API·DB 이름과 규칙이 정렬됐는가	field alignment·tests
DATA	parameter·transaction·constraint가 있는가	audit·DB tests·snapshot
ERROR	loading·empty·error와 4xx·5xx가 구분되는가	frontend/API tests
SAFETY	합성 data·local boundary·safe DOM인가	AGENTS·audit·CSP
REPRODUCE	exact start·test·reset이 작동하는가	generator·existing target exit 2
EVIDENCE	claim과 command·output·limit이 연결되는가	evidence 6 files

23.1. 최종 완료 문장

```
합성 관찰 1건을 화면에서 입력해 POST 201로 생성했고,
server allowlist·길이 검증과 parameterized SQLite transaction을 통과한 row가
response id와 최근 목록 첫 줄에 같은 값으로 나타났다.
18 tests, 13 audits, HTTP probe, DB snapshot으로 이를 재현했다.
인증·운영 배포·실제 data는 확인하지 않았다.
```

23.2. 다음 단계와의 경계

M08-02에서는 이 동일한 관찰 기능에 로그인과 권한을 추가합니다. 그때는 “누가 생성·조회할 수 있는가”, session·cookie·CSRF·authorization negative case가 새로운 vertical slice가 됩니다.

24. 셀프 테스트 12문항

1. Vertical slice의 완료 기준은 무엇인가요?

A. 화면 파일을 만들었다
B. API endpoint를 만들었다
C. 한 사용자 outcome이 UI→API→DB→UI를 왕복하고 증거가 있다
D. table을 만들었다

정답과 해설
정답은 C입니다. 각 층의 파일 존재가 아니라 같은 outcome과 값이 경계를 통과해 재현되는지가 기준입니다.

2. `required`와 `maxlength`가 있으면 server validation을 생략해도 되나요?

정답과 해설
안 됩니다. browser validation은 UX를 위한 빠른 피드백이며 우회할 수 있습니다. server가 신뢰 경계에서 type·allowlist·length를 다시 확인해야 합니다.

3. `fetch()`가 resolve됐으면 HTTP 요청은 성공한 것인가요?

정답과 해설
아닙니다. 404·422 같은 HTTP 오류도 response로 resolve될 수 있습니다. `response.ok`나 status를 확인해 application 실패를 throw해야 합니다.

4. `USABILITY`와 `사용성` 중 어느 값을 DB에 저장하나요? 왜인가요?

정답과 해설
stable code `USABILITY`를 저장합니다. 표시 문구 `사용성`은 번역·문구 변경 대상이므로 업무 key로 쓰면 UI 수정이 data contract 변경으로 번집니다.

5. Parameterized query가 server validation을 대신하나요?

정답과 해설
아닙니다. parameterization은 SQL 구조와 값을 분리합니다. validation은 category의 업무 허용값과 summary 길이처럼 값의 의미를 확인합니다.

6. `with connection:`이 connection까지 닫나요?

정답과 해설
Python sqlite3에서는 pending transaction을 성공 시 commit, exception 시 rollback하지만 connection을 닫지는 않습니다. 실습은 `closing(connect(...))`으로 resource close를 분리합니다.

7. 유효하지 않은 row INSERT가 실패한 뒤 무엇을 확인해야 하나요?

정답과 해설
오류 status·stable code뿐 아니라 DB row count가 변하지 않았는지 확인합니다. 이것이 partial effect가 없고 rollback됐다는 직접 증거입니다.

8. 왜 생성 response에 `id`와 `createdAt`을 넣나요?

정답과 해설
server·DB가 확정한 저장 결과를 client가 같은 값으로 표시하기 위해서입니다. client가 임의 번호와 시각을 만들면 화면과 DB가 달라질 수 있습니다.

9. `textContent`를 사용하는 이유는 무엇인가요?

정답과 해설
관찰 요약을 HTML 문법이 아니라 text node로 렌더하기 위해서입니다. 사용자 제어 문자열을 `innerHTML`에 넣는 불필요한 markup 실행 경계를 피합니다.

10. `reset_db.py`를 운영 DB에 써도 되나요?

정답과 해설
절대 안 됩니다. 이 script는 local 합성 DB와 sidecar를 삭제한 뒤 schema·seed를 재생성합니다. 운영 data에는 migration·backup·owner·복구 계획이 필요합니다.

11. 화면 screenshot 한 장이 증명하지 못하는 것을 두 가지 적으세요.

정답과 해설
예: DB constraint와 rollback, server-side validation, HTTP error contract, 다른 사용자의 동시 입력, authentication·authorization, 운영 복구. screenshot은 보이는 한 시점만 증명합니다.

12. 이번 MVP가 완료됐다는 최소 evidence 묶음은 무엇인가요?

정답과 해설
exact reset·test·audit·HTTP probe 결과, DB schema·row snapshot, runtime version, browser 저장 screenshot, non-goal·limit입니다. 모두 같은 생성물과 실행 상태를 가리켜야 합니다.

25. 현장 적용 Checklist

Outcome·범위

- [ ] actor·action·observable outcome을 한 문장으로 썼다
- [ ] 한 사용자 행동만 선택했다
- [ ] 로그인·외부 효과·운영 배포 등 non-goal을 적었다
- [ ] 실제 data 대신 합성 fixture를 쓴다

계약

- [ ] UI·API·DB field 이름과 변환 위치가 있다
- [ ] required·type·allowlist·length가 세 계약에서 충돌하지 않는다
- [ ] stable code와 display label을 분리했다
- [ ] 성공 status·body·Location을 적었다
- [ ] 400·413·415·422·404·405·500의 owner를 구분했다
- [ ] error.code와 field error가 안정적이다

Data·안전

- [ ] server에서 unknown field를 거부한다
- [ ] SQL value를 placeholder로 binding한다
- [ ] transaction commit·rollback을 test한다

- [ ] connection close와 transaction 수명을 분리한다
- [ ] NOT NULL·CHECK 등 DB constraint가 있다
- [ ] user-controlled text를 `textContent`로 렌더한다
- [ ] request body·개인정보·secret을 log하지 않는다

화면·증거

- [ ] loading·empty·ready·error 상태가 있다
- [ ] submit 중 중복 클릭을 줄인다
- [ ] 실패 때 입력을 보존하고 수정 위치를 알려 준다
- [ ] exact start·test·reset command가 있다
- [ ] behavior test·structure audit·HTTP probe가 있다
- [ ] DB snapshot·runtime version·browser screenshot이 있다
- [ ] evidence가 증명하지 않는 범위를 적었다
- [ ] 다른 사람이 새 폴더에서 처음부터 재현했다

26. 공식 출처와 적용 원칙

Python

- Python 3.12 sqlite3 — DB-API 2.0 interface: placeholder binding, transaction control, connection context manager, row factory의 기준으로 사용했습니다.
- Python 3.12 http.server: local HTTP 학습 server와 production 비권장 경계를 확인했습니다.
- Python contextlib.closing: SQLite connection resource를 명시적으로 닫는 패턴에 적용했습니다.
- Python http.HTTPStatus: 의미 있는 status code 상수를 사용했습니다.

SQLite

- SQLite CREATE TABLE: `PRIMARY KEY`, `NOT NULL`, `CHECK` constraint의 기준으로 사용했습니다.
- SQLite Transactions: BEGIN·COMMIT·ROLLBACK과 한 writer 원칙을 확인했습니다.
- SQLite Query Language: CREATE INDEX: 최신순 query와 index 의도를 정렬했습니다.
- SQLite Datatypes: dynamic typing과 `typeof` guard의 한계를 확인했습니다.

Browser·HTTP

- MDN Fetch API: request·response 비동기 흐름의 기준으로 사용했습니다.

- MDN Using the Fetch API: POST JSON과 non-2xx에서 `response.ok` 확인을 적용했습니다.
- MDN Constraint validation: client validation이 server validation을 대신하지 않는 경계를 확인했습니다.
- MDN Node.textContent: 사용자 문자열을 text로 렌더하는 기준으로 사용했습니다.
- MDN HTTP response status codes: 2xx·4xx·5xx 의미를 확인했습니다.
- MDN Content Security Policy: local same-origin CSP의 학습 기준으로 사용했습니다.

Secure development

- OWASP Input Validation Cheat Sheet: early server validation, syntactic·semantic validation, allowlist를 적용했습니다.
- OWASP SQL Injection Prevention Cheat Sheet: parameterized query를 문자열 조립보다 우선했습니다.
- OWASP REST Security Cheat Sheet: method·content type·status·generic error의 API 경계를 검토했습니다.
- OWASP Cross Site Scripting Prevention Cheat Sheet: context-aware safe output 원칙을 적용했습니다.

이 교재의 적용 원칙

1. 공식 문서가 말하는 API 동작과 이 실습의 설계 결정을 구분했습니다.
2. `http.server`를 package 없는 local 학습에만 사용하고 production 적합성으로 확대 해석하지 않았습니다.
3. browser validation·server validation·DB constraint를 서로 대체하지 않고 역할별로 겹쳤습니다.
4. parameterized query를 사용하면서도 업무 allowlist와 authorization 필요성을 별도로 남겼습니다.
5. screenshot·test·probe·DB snapshot의 증명 범위와 한계를 함께 기록했습니다.
6. authentication·authorization·운영 배포는 다음 slice로 미루고 이번 완료 판정에 섞지 않았습니다.

다음 매뉴얼: M08-02「로그인과 권한이 있는 기능 만들기」