M08-02 · GIBALJA VISUAL MANUAL

로그인과 권한이 있는 기능 만들기

> **한 문장 목표:** `자격 정보 검증 → 임의 세션 발급 → HttpOnly cookie → 매 요청 세션·CSRF·역할 검증 → 허용된 행동만 실행 → logout 폐기`를 한 번 연결하고, 성공보다 실패 증거로 보호 경계를 확인합니다.

난이도	개념	실습	셀프 테스트	최종 산출물
Level 2	70분	65분	15분	역할별 화면, 보호 API, session·CSRF·permission 증거 packet

로그인 화면을 만드는 것과 로그인된 사용자만 행동하게 만드는 것은 다릅니다. 이번 실습은 **화면의 로그인 모양**이 아니라 **서버가 매 요청에서 신원과 행동 권한을 다시 판단하는 방법**을 다룹니다. 계정·이름·기록은 모두 합성 데이터입니다.

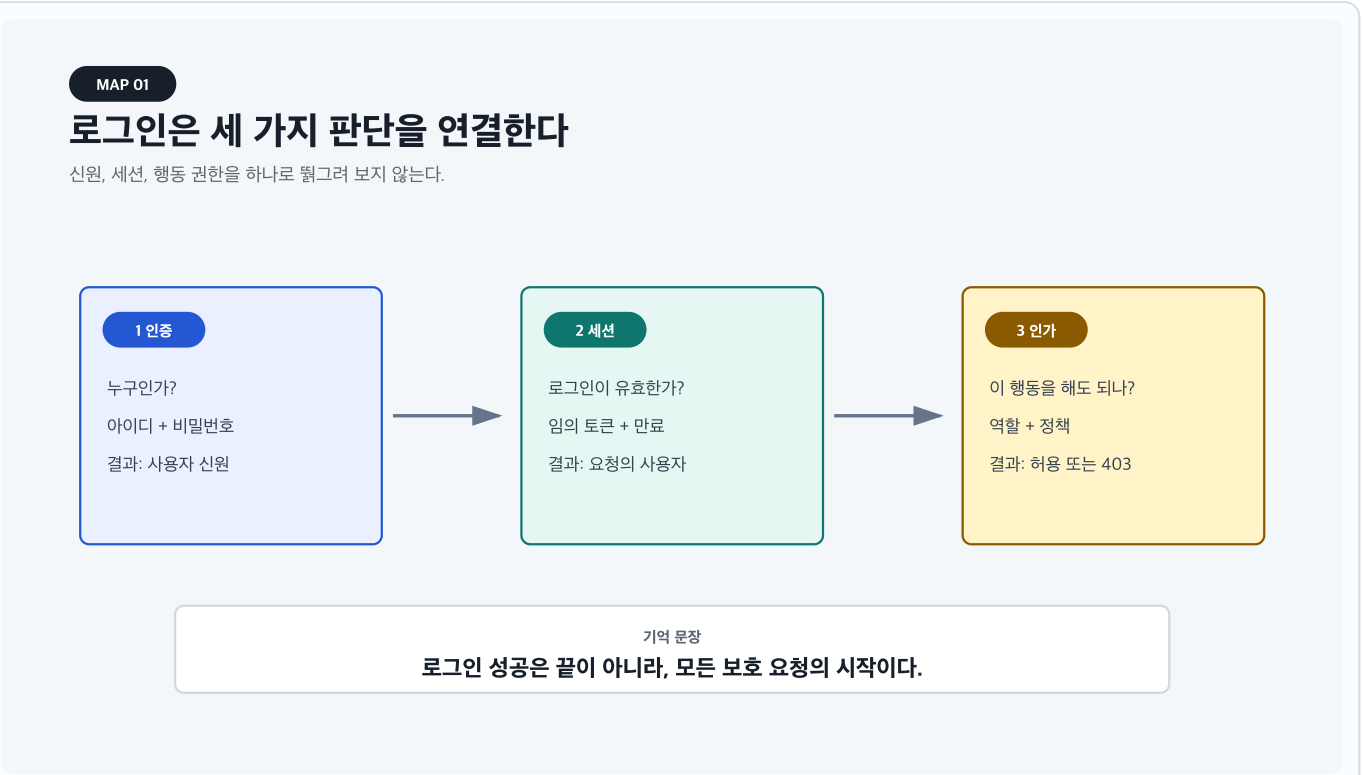
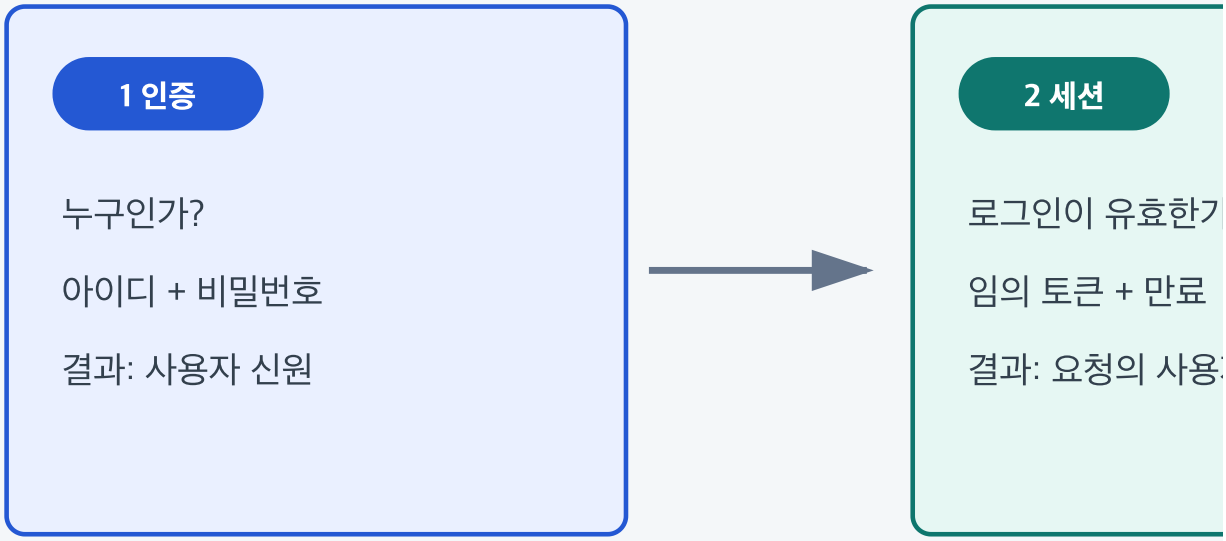


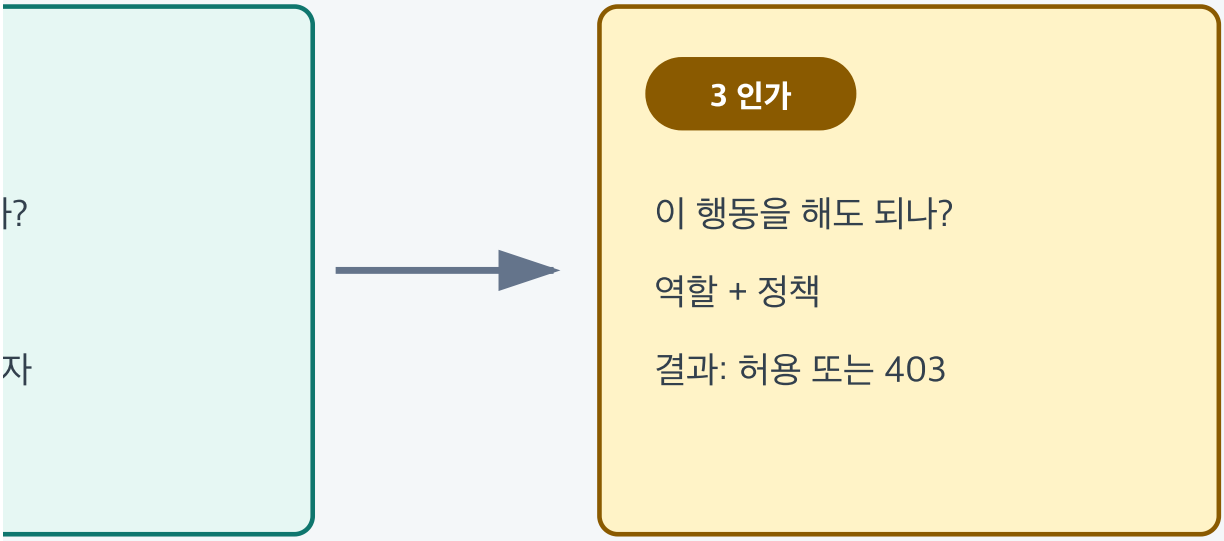
그림 1. 인증은 누구인지, 세션은 그 확인이 아직 유효한지, 인가는 그 사용자가 이 행동을 해도 되는지를 판단합니다.

신원, 세션, 행동 권한을 하나로 뭉뚱그려 보지 않는다.



기억

로그인 성공은 끝이 아니라,



문장

모든 보호 요청의 시작이다.

0. 이 PDF를 공부하는 방법

1회차 · 그림만 읽기 · 25분

그림 1부터 그림 16까지 제목과 하단 결론만 보세요. 다음 여덟 문장을 소리 내어 말할 수 있으면 1회차는 완료입니다.

```
인증은 누구인지, 인가는 해도 되는지를 묻는다.
로그인 성공은 세션 생명주기의 시작이다.
브라우저는 임의 세션 토큰만, 서버는 역할과 만료를 보관한다.
상태 변경은 세션·CSRF·권한을 모두 통과해야 한다.
역할과 행동이 정책에 없으면 기본은 거부다.
401은 다시 로그인, 403은 권한이나 행동을 바꿀 신호다.
화면은 권한을 설명하고, 서버는 권한을 강제한다.
보안은 성공 한 번이 아니라 실패 여섯 개로 증명한다.
```

2회차 · 두 계정 비교 · 15분

실습 생성기를 실행합니다.

```
./02_Labs/G08_Fullstack/L08-02_create-auth-permission-practice.sh
```

생성기는 외부 package와 network 없이 다음을 만듭니다.

```
gibalja-auth-permission-practice/
├── app/       합성 계정·세션·권한이 있는 local 앱
└── evidence/  reset·test·audit·contract probe·storage·version 증거
```

두 계정을 각각 실행합니다.

계정	아이디	실습 비밀번호	예상 행동
기획자	`planner`	`Planner-Only-2026!`	목록 조회 + 관찰 작성
조회자	`viewer`	`Viewer-Only-2026!`	목록 조회만

3회차 · 실패 여정 추적 · 35분

성공 저장보다 다음 실패를 먼저 읽습니다.

```
익명 GET 보호 목록 = 401
없는 계정 login = 401
틀린 비밀번호 login = 401, 같은 message
viewer GET 목록 = 200
viewer POST 작성 = 403
planner POST, CSRF 없음 = 403
planner POST, CSRF 일치 = 201
logout 후 GET 목록 = 401
```

4회차 · 내 기능에 적용 · 75분

로그인·권한 검증 기록 양식을 채우고 단계별 실습서를 따릅니다. 낯선 표현은 인증·세션·인가 용어집에서 찾습니다.

1. 이번 Vertical Slice의 결과와 비목표를 고정합니다

M08-01의 관찰 보드는 누구나 목록을 읽고 쓸 수 있었습니다. M08-02에서는 같은 화면과 데이터에 신원과 행동 정책을 추가합니다.

1.1. 검증할 outcome

```
actors:
  - planner: 보호된 목록을 읽고 합성 관찰을 작성
  - viewer: 보호된 목록만 읽음
starting_state:
  - 합성 계정 2개
  - 합성 관찰 2건
  - 활성 session 0개
success_evidence:
  - planner POST /api/observations = 201
  - viewer POST /api/observations = 403
  - anonymous GET /api/observations = 401
  - logout 후 같은 보호 GET = 401
```

1.2. 의도적 non-goal

제외	이유	운영으로 갈 때
회원가입·비밀번호 재설정	identity lifecycle이 급격히 커짐	검증된 identity provider·framework 사용
MFA	로컬 역할 흐름에 집중	위험 기반 step-up·MFA 설계
OAuth·OIDC·SSO	제3자 인증 계약 제외	mature IdP와 redirect·state·nonce 검증
TLS 종단	loopback HTTP 학습 제한	전 로그인·인증 page HTTPS
login throttling·lockout	분산 상태·운영 정책 제외	rate limit·monitoring·safe recovery
영구 audit log	민감 정보·보존 정책 제외	tamper-resistant audit·retention
실제 사용자 data	학습 안전 경계	동의·최소화·보존·파기 정책

중요한 경계
이 실습의 Python `http.server`는 학습용 loopback server입니다. 인터넷에 공개하지 않고, 실제 계정·비밀번호·사업 데이터를 입력하지 않습니다.

2. 인증·세션·인가를 서로 다른 판단으로 분리합니다

그림 2. 인증은 identity를, 인가는 action permission을 판단합니다. 두 실패를 같은 오류로 취급하면 사용자의 다음 행동을 안내할 수 없습니다.

401과 403을 구분하면 장애 위치와 다음 행동이 보인다.

인증 AUTHN

질문

너는 누구인가?

입력: 자격 정보, 세션 토큰

실패: 401 Unauthorized

다음: 로그인 또는 세션 갱신

인가 AUTHZ

질문

너는 이것을 해도 되나?

입력: 신원, 역할, 자원, 행동

실패: 403 Forbidden

다음: 권한 요청 또는 행동 변경

2.1. 세 책임의 입력과 출력

책임	질문	주요 입력	성공 출력	실패
authentication	이 자격 정보가 알려진 사용자인가	username, password	user identity	401
session resolution	이 request가 아직 유효한 login과 연결되는가	opaque cookie token	user, role, expiry, CSRF	401 또는 signed-out state
authorization	이 user role이 이 action을 해도 되는가	role, action, resource	allow	403

2.2. 화면 로그인 상태를 신뢰할 수 없는 이유

브라우저에서 다음과 같은 플래그를 바꾼는 것은 쉽습니다.

```
state.user = { role: "planner" };
form.hidden = false;
```

그러나 이 변경은 server session도, DB permission도 바꾸지 않습니다. 화면의 역할 표시는 사용 경험을 돕는 표현입니다. 보안 판단은 server가 다시 해야 합니다.

> **30초 확인 1**
> viewer가 작성 버튼을 개발자 도구로 다시 보이게 만들었습니다. 이 사용자가 작성할 수 없는 최종 이유는 버튼이 아니라 어디에 있어야 하나요?

3. 역할과 행동을 Permission Matrix로 고정합니다

역할 이름만 적으면 사람마다 다르게 해석합니다. 역할을 행동 집합으로 펼치면 테스트 가능한 정책이 됩니다.

POLICY 09

권한 정책은 역할과 행동의 표로 고정한다

명시적으로 허용한 셀만 통과하고, 모르는 역할과 행동은 기본 거부한다.

역할 / 행동	목록 조회	관찰 작성	관리 행동
기획자 planner	허용	허용	거부
조회자 viewer	허용	거부	거부
미지 역할	거부	거부	거부

DENY BY DEFAULT · 정책에 없으면 권한도 없다

그림 3. 허용은 명시하고 나머지는 거부합니다. 미지 역할이 실수로 권한을 얻지 못합니다.

명시적으로 허용한 셀만 통과하고, 모르는 역할과 행동은 기본 거부한다.

역할 / 행동	목록 조회
기획자 planner	허용
조회자 viewer	허용
미지 역할	거부

DENY BY DEFAULT · 정

	관찰 작성	관리 행동
	허용	거부
	거부	거부
	거부	거부

3.1. 코드로 표현한 정책

```
PERMISSIONS = {
    "planner": frozenset({"observation:read", "observation:create"}),
    "viewer": frozenset({"observation:read"}),
}

def is_allowed(role, action):
    return action in PERMISSIONS.get(role or "", frozenset())
```

`PERMISSIONS.get(..., frozenset())`의 기본값이 빈 집합입니다. 정책에 없는 role은 자동으로 모든 action을 거부합니다.

3.2. 이름보다 행동을 검색합니다

나쁜 질문	더 좋은 질문
이사는 무엇이든 할 수 있는가	이 role이 `invoice:approve` action을 할 수 있는가
관리자니까 허용하자	정책에 이 action이 명시되었는가
버튼이 보이니 허용된다	server가 user·action·resource를 검증했는가

4. 로그인 요청을 Password에서 Session으로 전환합니다

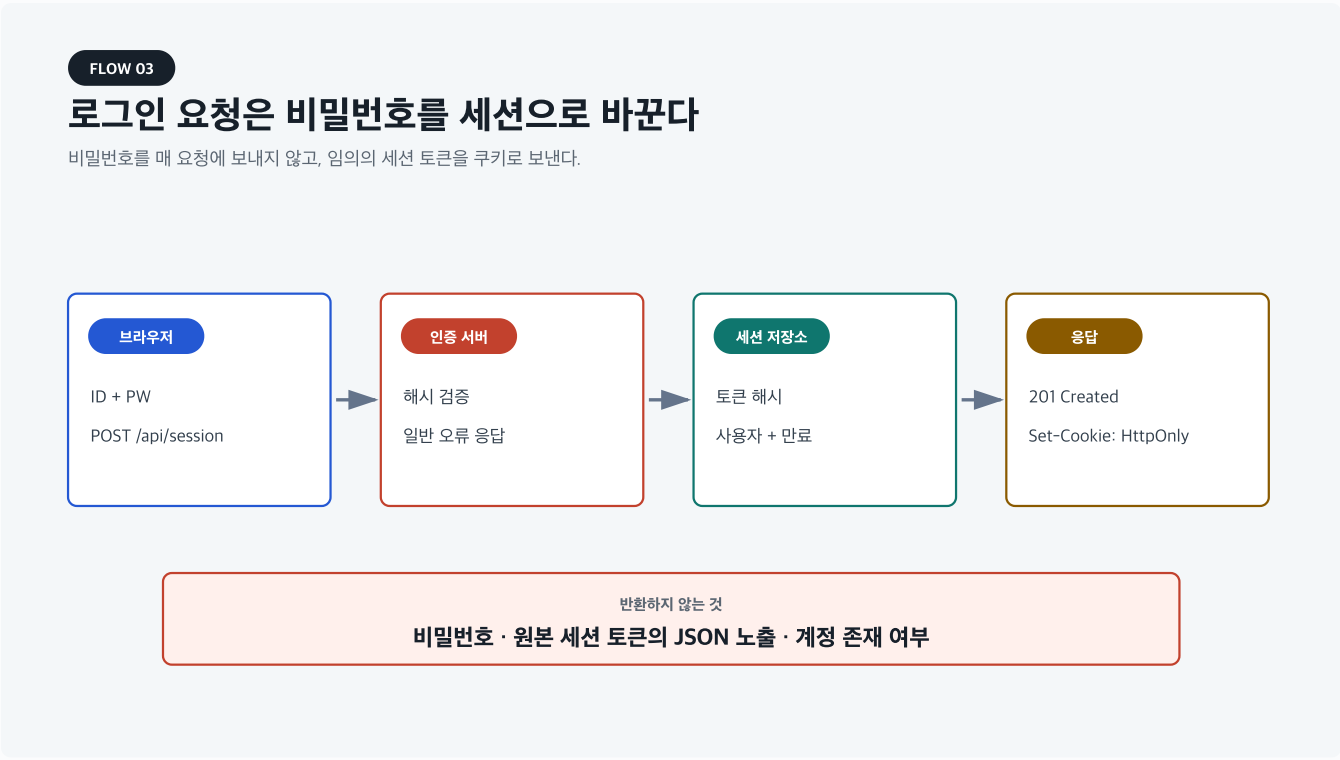
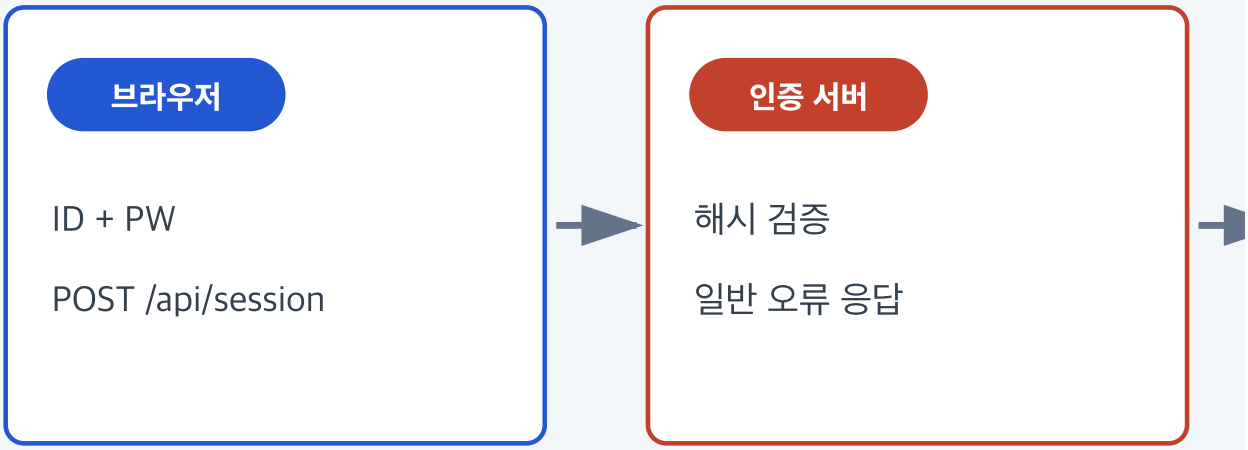


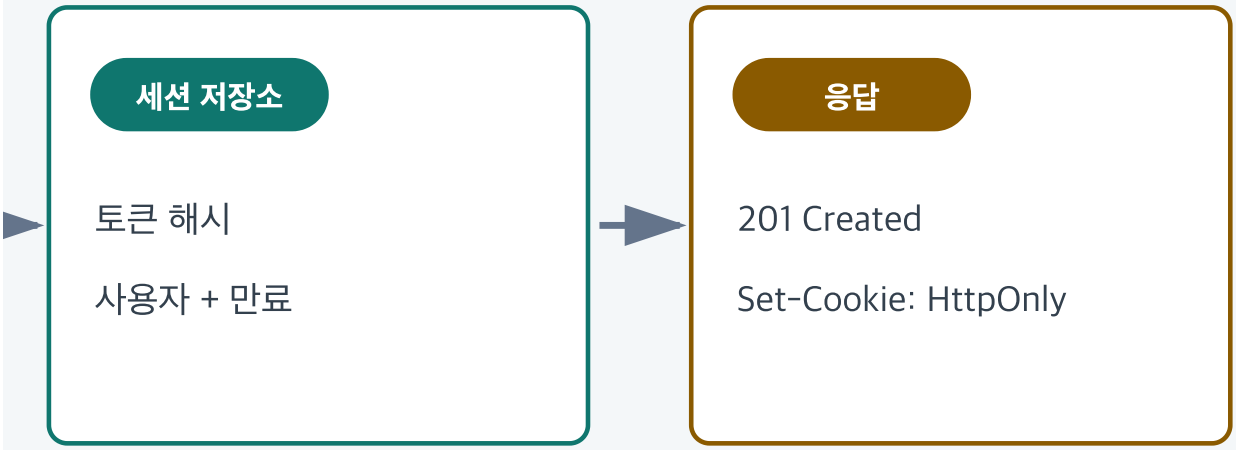
그림 4. 로그인은 비밀번호를 매 요청에 보내는 구조가 아니라, 추측하기 어려운 임의 session token으로 바꾸는 교환입니다.

비밀번호를 매 요청에 보내지 않고, 임의의 세션 토큰을 쿠키로 보낸다.



반환하지

비밀번호 · 원본 세션 토큰의



| 않는 것

JSON 노출 · 계정 존재 여부

4.1. Endpoint 계약

```
POST /api/session
Content-Type: application/json

{"username":"planner","password":"Planner-Only-2026!"}
```

성공:

```
HTTP/1.1 201 Created
Set-Cookie: session_id=<opaque>; Path=/; HttpOnly; SameSite=Lax; Max-Age=900
Cache-Control: no-store
Content-Type: application/json

{"data":{"user":{"username":"planner","role":"planner"},
         "csrfToken":"<separate-random-value>",
         "expiresAt":"<UTC-time>"}}
```

실패:

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json

{"error":{"code":"INVALID_CREDENTIALS",
          "message":"아이디 또는 비밀번호를 확인하세요."}}
```

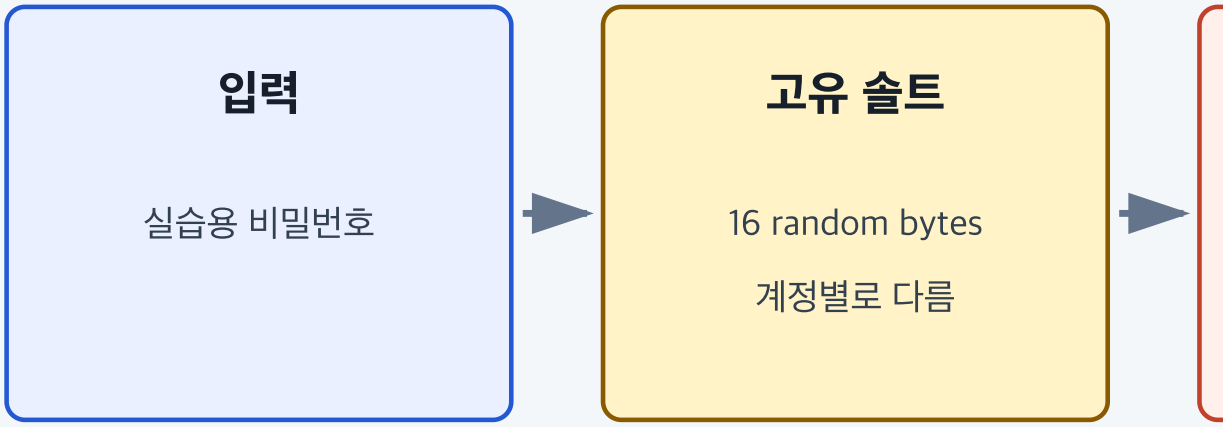
4.2. 성공 응답에서 제외할 것

- password 원문
- password hash·salt·iteration
- raw session token의 JSON field
- 계정이 실제로 존재하는지 알려 주는 세부 오류
- 필요 없는 개인정보·내부 식별자

5. Password를 원문이 아닌 검증 자료로 저장합니다

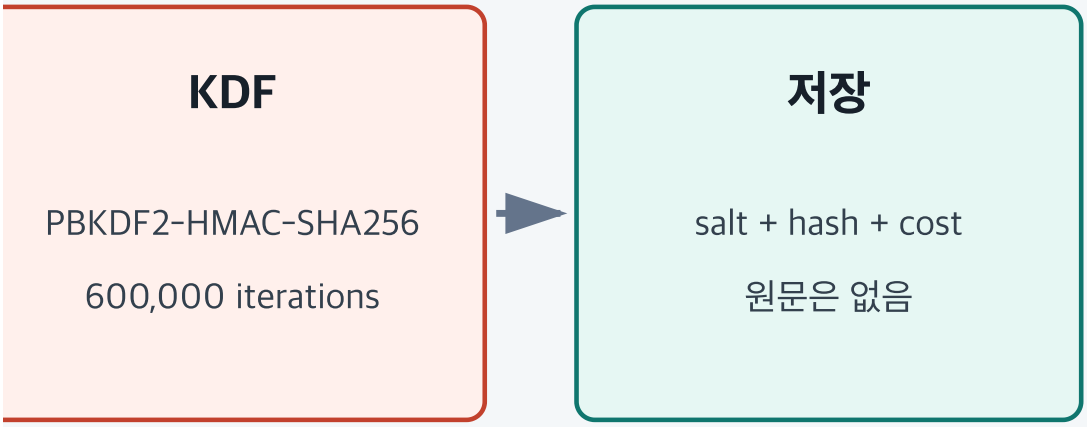
그림 5. 비밀번호 저장의 목적은 나중에 읽는 것이 아니라, 후보 비밀번호가 같은지 비교하는 것입니다.

원문을 저장하지 않고, 고유 솔트와 작업 비용이 있는 KDF를 쓴다.



실습 선택

표준 라이브러리로 과정을 보여 주기 위해 PBKDF2 600k를 사용



운영 선택

신규 시스템은 Argon2id를 우선 검토

5.1. 학습 구현의 저장 필드

필드	예	이유
`password_salt`	계정별 random 16 bytes	같은 password도 다른 hash
`password_hash`	PBKDF2-HMAC-SHA256 32 bytes	원문 대신 검증 결과
`password_iterations`	600000	작업 비용 기록·향후 상향

```
hashlib.pbkdf2_hmac(
    "sha256",
    password.encode("utf-8"),
    salt,
    600_000,
)
```

5.2. 왜 일반 SHA-256 한 번이 아닌가

비밀번호는 사전 대입 공격의 대상입니다. 빠른 일반 hash는 공격자에게도 빠릅니다. password KDF는 salt와 의도적인 작업 비용으로 추측 1회의 비용을 높입니다.

5.3. 실습 선택과 운영 선택

OWASP Password Storage Cheat Sheet는 신규 시스템에 Argon2id를 우선 권고합니다. 이 실습은 외부 package 없이 Python 표준 라이브러리로 salt·cost·KDF를 눈으로 보기 위해 PBKDF2-HMAC-SHA256 600,000회를 사용합니다. 이 선택을 모든 운영 시스템에 그대로 복사하지 않습니다.

5.4. Constant-time 비교

```
hmac.compare_digest(actual_hash, expected_hash)
```

일반 문자열 비교는 중간에 틀리면 일찍 종료할 수 있습니다. `compare_digest`는 비교 시간 차이로 정보가 새는 위험을 줄이는 도구입니다.

6. Login 오류는 계정 존재 여부를 노출하지 않습니다

다음 두 장면을 같은 status·code·message로 돌려줍니다.

```
존재하지 않는 username + 아무 password
존재하는 username + 틀린 password
```

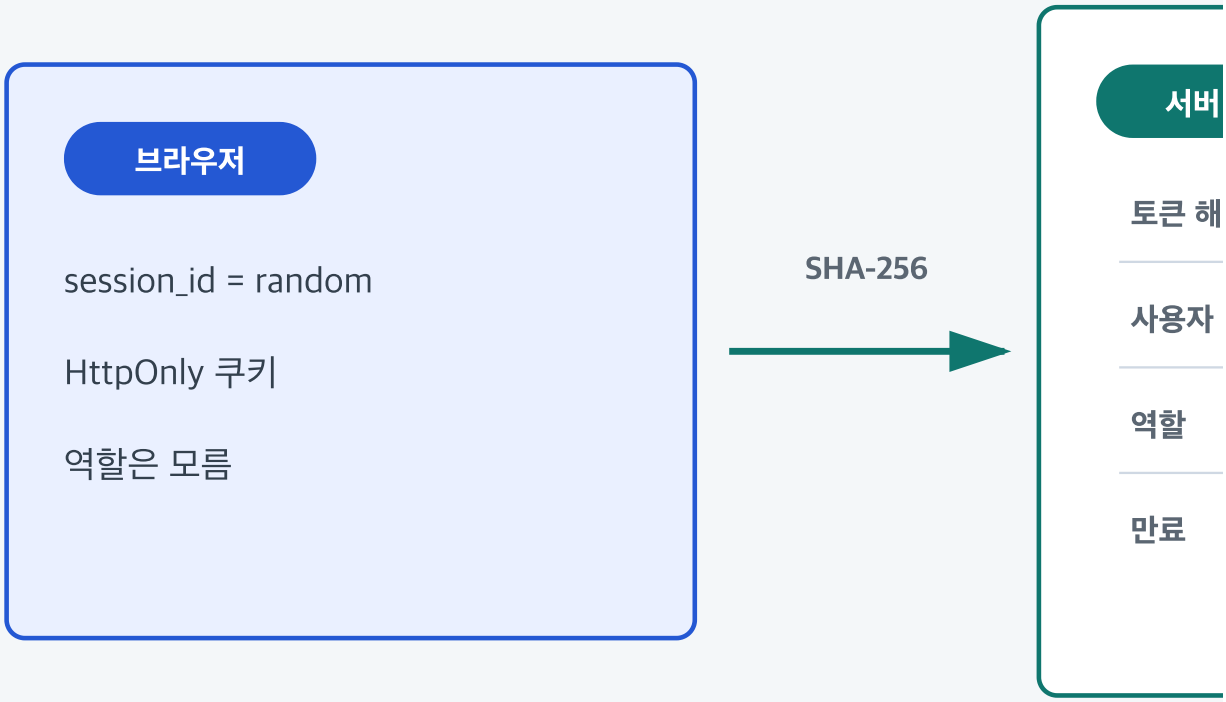
지나치게 자세한 오류	문제
`planner 계정은 존재하지만 password가 틀렸습니다`	계정 목록 확인 가능
`viewer 계정은 없습니다`	account enumeration
`비밀번호 3번째 문자까지 일치`	필요 없는 검증 정보 노출

실습 코드는 없는 계정에도 dummy salt·hash로 KDF 작업을 수행합니다. 그러나 이 로컬 구현이 시간 사이드 채널을 완전히 제거한다고 간주하지 않습니다. 운영은 인증 framework, throttling, monitoring, MFA를 함께 설계합니다.

7. Opaque Server-side Session으로 Request를 User에 연결합니다

그림 6. 브라우저는 의미 없는 임의 토큰을, server는 그 토큰의 hash와 user·role·expiry를 보관합니다.

브라우저는 추측하기 어려운 임의 값만 보관하고 역할과 만료는 서버에서 읽는다.



DB가 털려도 원본 쿠키 토큰을

세션 표

시	32 bytes
	planner
	planner
	+15 minutes

바로 재사용하지 못하게 한다.

7.1. Raw token과 DB token hash

```
raw_token = secrets.token_urlsafe(32)
token_hash = hashlib.sha256(raw_token.encode("ascii")).digest()
```

위치	보관하는 것	보관하지 않는 것
browser cookie jar	raw opaque token	role·password·CSRF token
JavaScript memory	CSRF token, user display data	raw session token
server sessions table	token hash, user id, CSRF token, times	raw session token·password

DB snapshot이 노출되었을 때 raw cookie token을 바로 얻지 못하게 하는 방어입니다. 그러나 세션 DB 유출도 중대 사고이므로 접근 제어·암호화·폐기 절차가 필요합니다.

7.2. Session resolution

```
Cookie header parse
→ raw token의 길이·형식 방어
→ SHA-256 lookup hash
→ sessions JOIN users
→ account active 확인
→ expires_at 확인
→ user·role·csrfToken·expiresAt 반환
```

8. Cookie 속성으로 Browser의 전송 규칙을 설계합니다

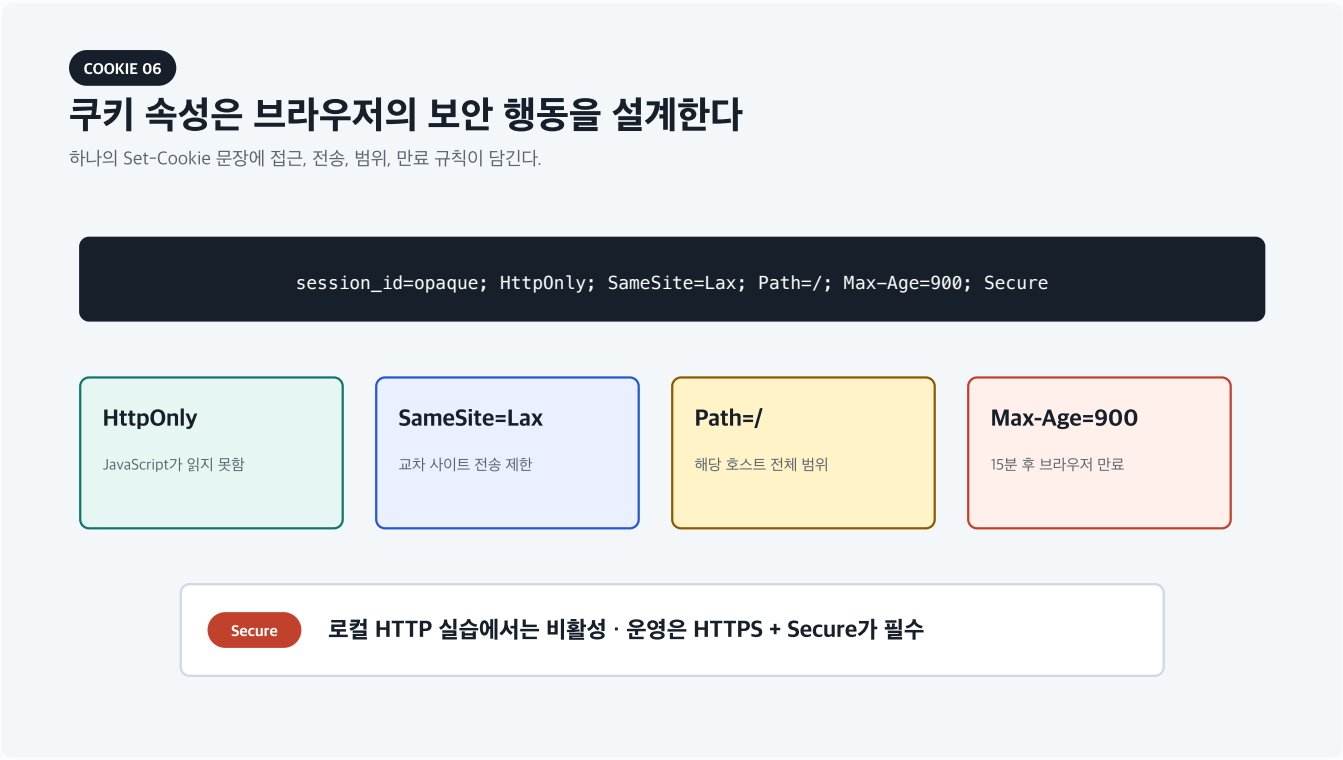


그림 7. Cookie attribute는 장식이 아니라 browser에게 접근·전송·범위·만료 규칙을 주는 계약입니다.

하나의 Set-Cookie 문장에 접근, 전송, 범위, 만료 규칙이 담긴다.

```
session_id=opaque; HttpOnly; SameSi
```

HttpOnly

JavaScript가 읽지 못함

SameSite=Lax

교차 사이트 전송 제한

Secure

로컬 HTTP 실습에서는 비활성 · 운영

```
te=Lax; Path=/; Max-Age=900; Secure
```

Path=/

해당 호스트 전체 범위

Max-Age=900

15분 후 브라우저 만료

은 HTTPS + Secure가 필수

속성	실습 값	효과	한계
`HttpOnly`	설정	JavaScript `document.cookie`로 세션 토큰 읽기 제한	XSS 자체를 없애지 않음
`SameSite=Lax`	설정	일부 cross-site 요청의 cookie 전송 제한	CSRF token을 대체하지 않음
`Path=/`	설정	해당 host의 전체 path에 전송	authorization 경계가 아님
`Max-Age=900`	설정	browser 측 15분 만료	server expiry도 따로 검증해야 함
`Secure`	local HTTP에서 비활성	HTTPS에서만 전송	운영에서 필수

PRODUCTION BOUNDARY

로그인 page와 모든 authenticated page는 HTTPS로만 제공하고 session cookie에 Secure를 설정합니다.

9. Session에 발급·회전·만료·폐기 생명주기를 부여합니다

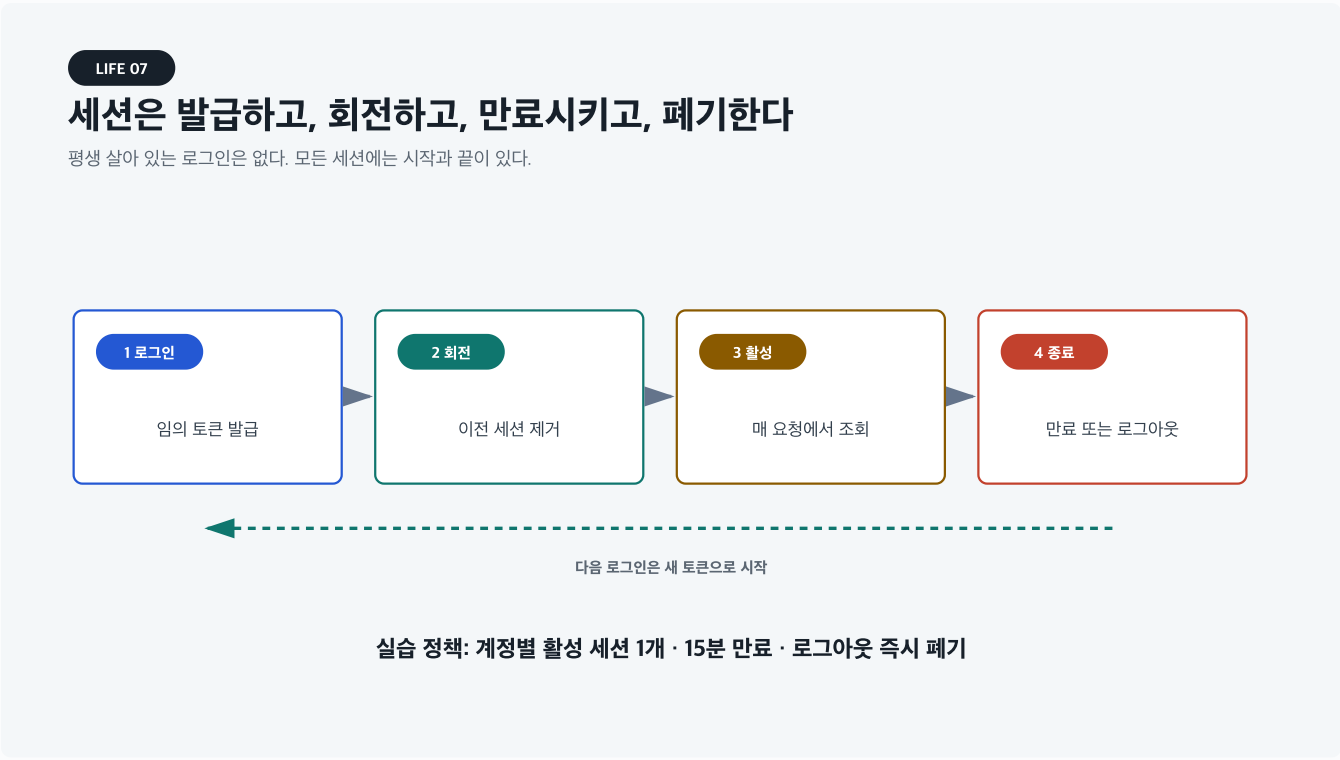
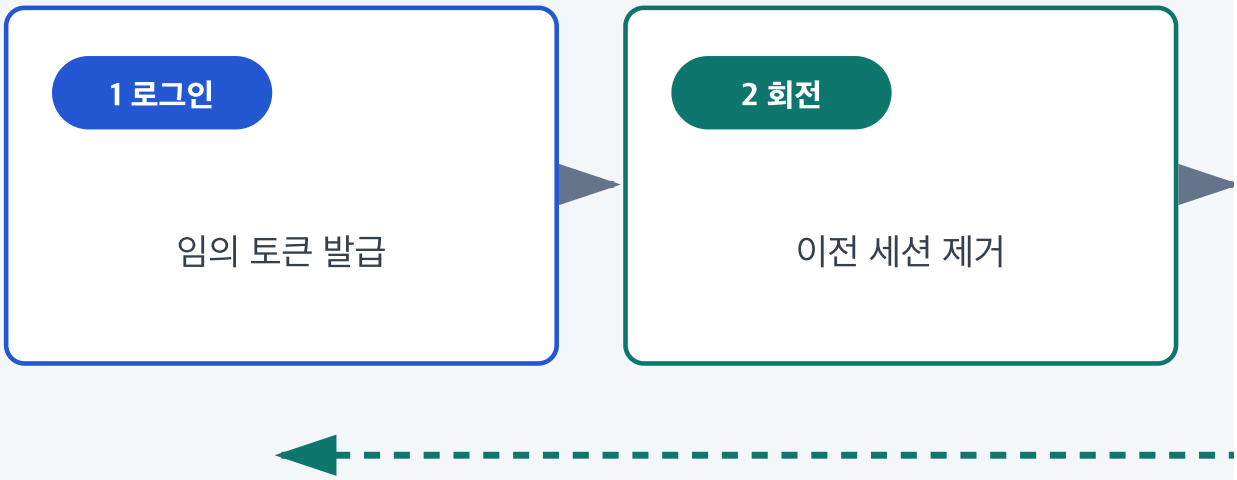


그림 8. 로그인 성공은 영구 권한이 아닙니다. 이 실습은 계정별 활성 session 1개, 15분 expiry, logout 즉시 revocation을 사용합니다.

평생 살아 있는 로그인은 없다. 모든 세션에는 시작과 끝이 있다.



실습 정책: 계정별 활성 세션 1개

새 토큰으로 시작

· 15분 만료 · 로그아웃 즉시 폐기

9.1. Login rotation

로그인할 때 같은 user의 이전 session을 삭제하고 새 token을 발급합니다.

```
DELETE FROM sessions WHERE user_id = ?;
INSERT INTO sessions
  (token_hash, user_id, csrf_token, created_at, expires_at)
VALUES (?, ?, ?, ?, ?);
```

이것은 학습용 단순 정책입니다. 운영 서비스의 다중 device session, session list, remote logout 정책은 별도로 설계해야 합니다.

9.2. Server-side expiry

Browser cookie가 남아 있어도 server `expires_at`이 지났으면 거부합니다. 만료 row는 조회 시 삭제합니다.

9.3. Logout revocation

```
DELETE /api/session
X-CSRF-Token: <current-token>
Cookie: session_id=<opaque>
```

Server는 session row를 삭제하고 browser에 `Max-Age=0`을 보냅니다. 두 측을 모두 정리해야 합니다.

10. Session 조회와 Protected Resource 요청을 구분합니다

첫 화면은 로그인 여부를 알아야 합니다. 이 실습은 세션 조회에서 `signed out`을 정상 UI state로 돌려줍니다.

```
GET /api/session

200 OK
{"data":null}
```

반면 보호된 자원은 미인증을 401로 돌려줍니다.

```
GET /api/observations

401 Unauthorized
{"error":{"code":"AUTHENTICATION_REQUIRED","message":"로그인이 필요합니다."}}
```

이 구분으로 첫 page load에 예상된 401 console noise를 남기지 않으면서, 실제 보호 resource의 인증 계약은 유지합니다.

11. CSRF Token으로 Cookie만 있는 State Change를 막습니다

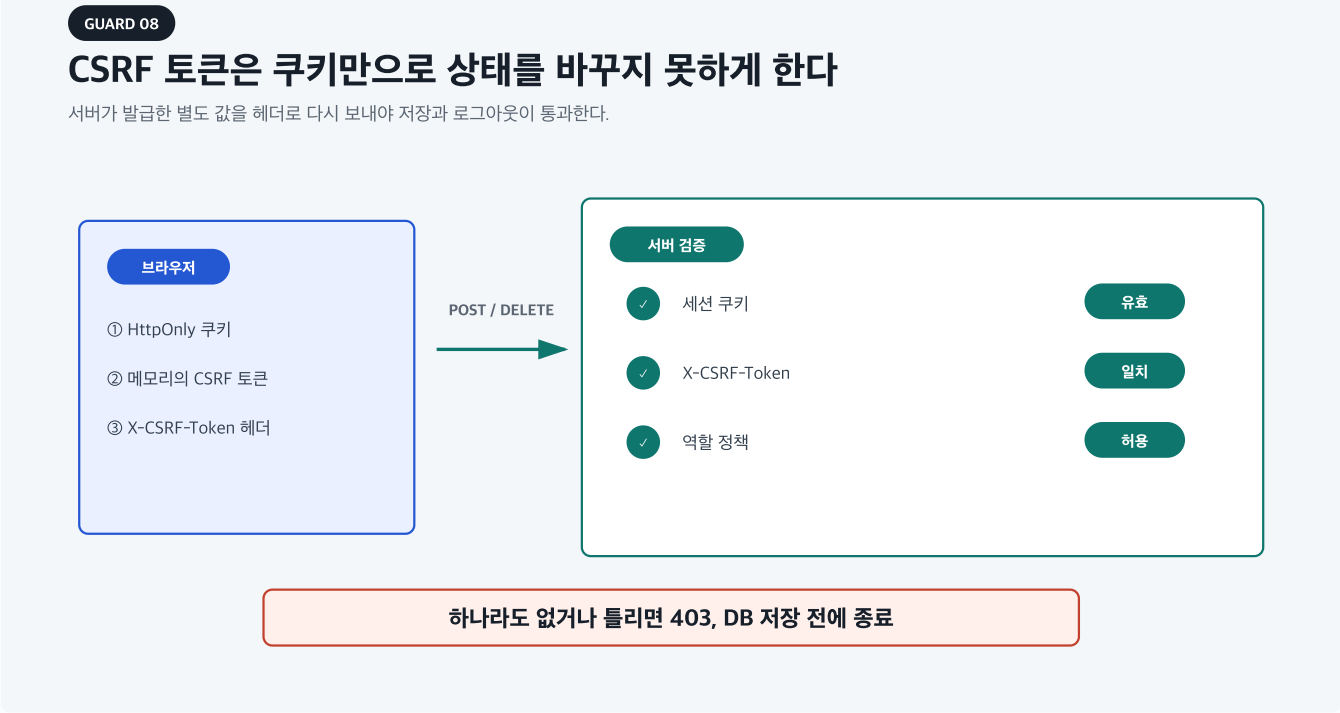
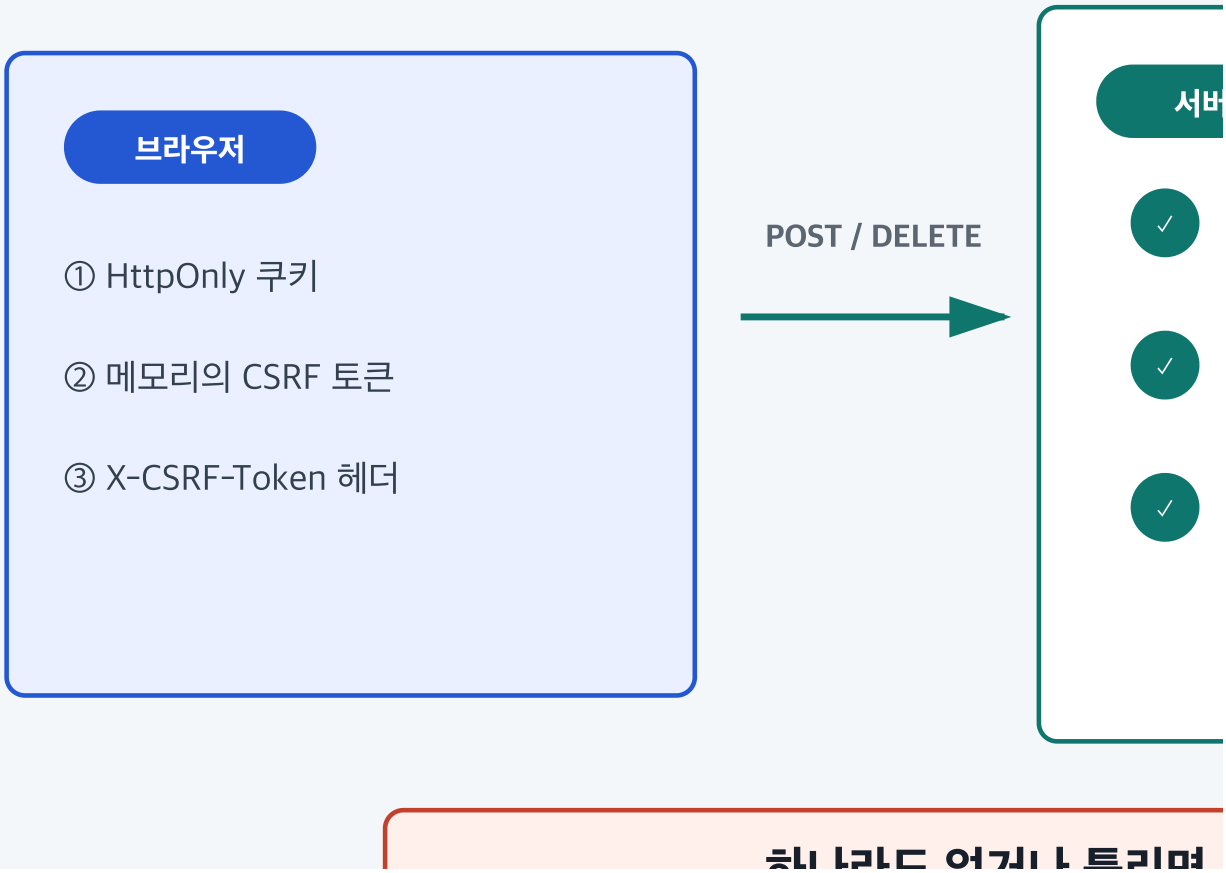


그림 9. 작성과 logout은 유효한 session cookie와 일치하는 `X-CSRF-Token` 헤더를 모두 요구합니다.

서버가 발급한 별도 값을 헤더로 다시 보내야 저장과 로그아웃이 통과한다.



검증

세션 쿠키	유효
X-CSRF-Token	일치
역할 정책	허용

403 DB 저장 전에 종료

11.1. 실습의 Synchronizer Token 흐름

```
login 성공
→ server가 session 전용 CSRF token 발급
→ JSON response로 CSRF token 전달
→ JavaScript memory에만 보관
→ POST / DELETE의 X-CSRF-Token 헤더로 전송
→ server가 session row의 token과 constant-time 비교
```

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

11.2. SameSite만으로 끝내지 않는 이유

`SameSite=Lax`는 유용한 defense-in-depth지만 모든 CSRF 상황을 단독으로 해결하지 않습니다. 상태 변경 요청은 별도 token과 origin·framework 방어를 함께 검토합니다.

11.3. 이 실습의 Login CSRF 한계

Pre-authentication login request에는 아직 session synchronizer token이 없습니다. 실습의 `SameSite=Lax`만으로 login CSRF와 모든 cross-origin 위협을 완전히 해결했다고 간주하지 않습니다. 운영은 mature identity framework의 login CSRF 방어, origin 검증, IdP 계약을 적용합니다.

12. Protected Write의 Gate 순서를 고정합니다

그림 10. 신원·요청 의도·행동 권한을 확인한 뒤에만 input을 해석하고 DB를 바꿉니다.

실패는 빨리 종료하고, 모든 문을 통과한 요청만 DB에 도달한다.

서버가 매 요청(

화면에서 버튼을 숨겨도 API

에서 재검증한다

권한 검증은 생략할 수 없다.

```
1. session cookie를 해석하고 유효한 session인지 확인  → 실패 401
2. X-CSRF-Token이 session token과 일치하는지 확인 → 실패 403
3. role이 observation:create를 가지는지 확인      → 실패 403
4. JSON·field·business input을 검증                    → 실패 4xx/422
5. parameterized query로 transaction 저장               → 실패 rollback/5xx
6. 201·Location·새 row를 반환
```

이 순서의 중요한 점은 권한 없는 사용자의 요청이 DB에 도달하지 않는다는 것입니다.

30초 확인 2
viewer가 잘못된 JSON으로 작성을 요청했습니다. 이 실습의 순서에서 먼저 보여야 하는 오류는 permission 403입니까, input 422입니까? 왜 그런가요?

13. 401과 403을 다음 행동으로 읽습니다

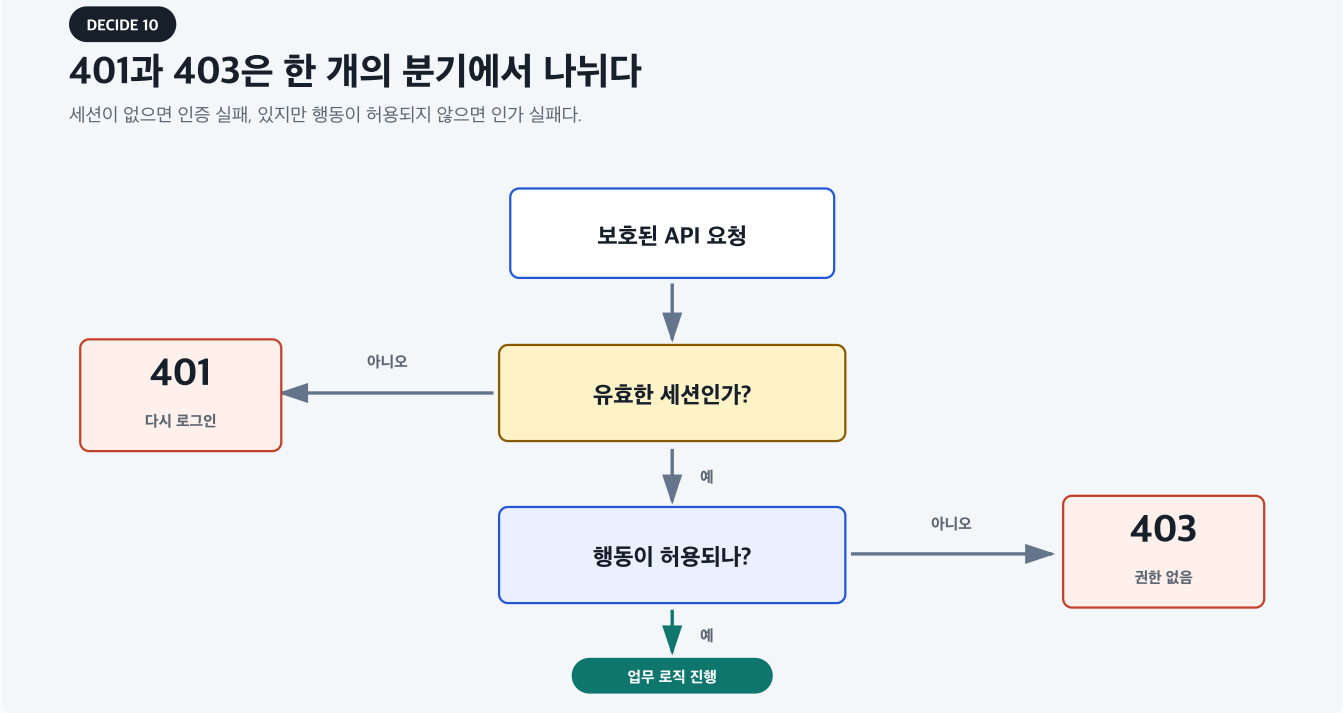
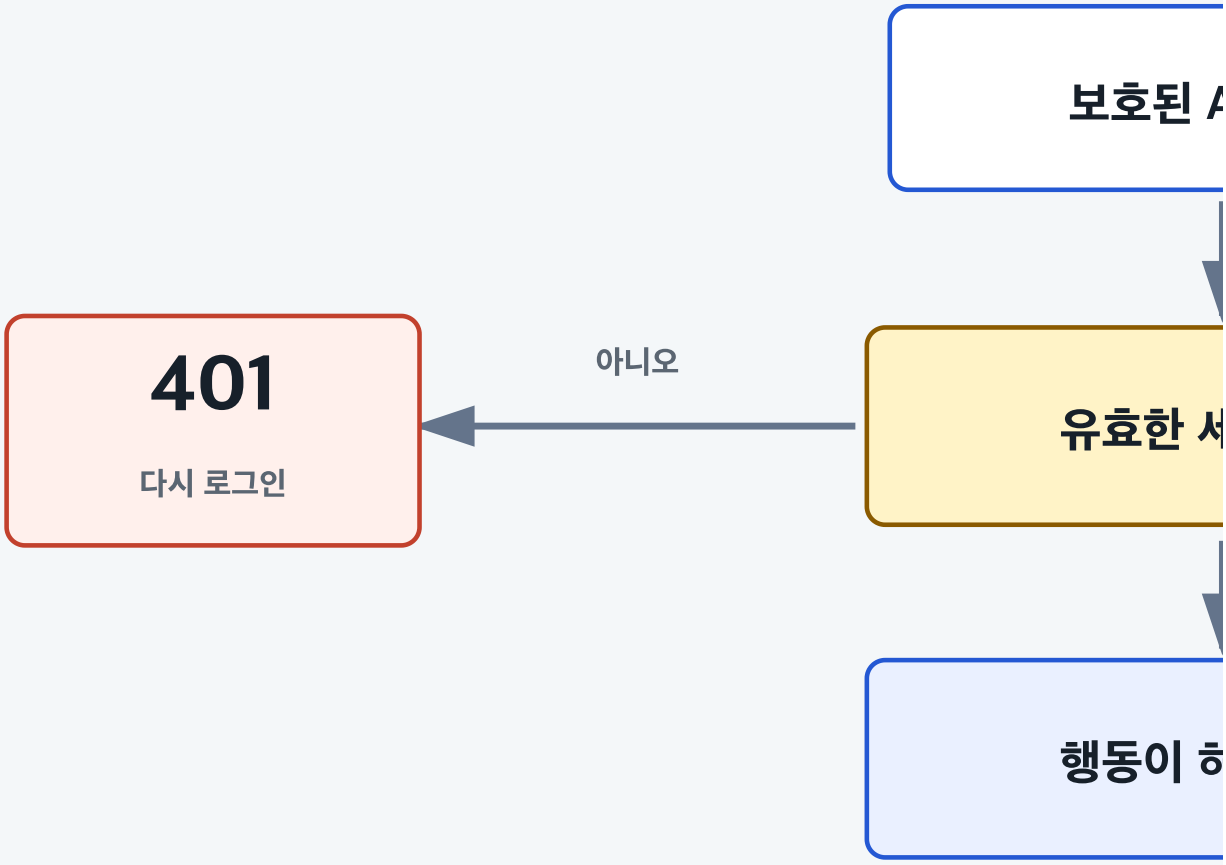
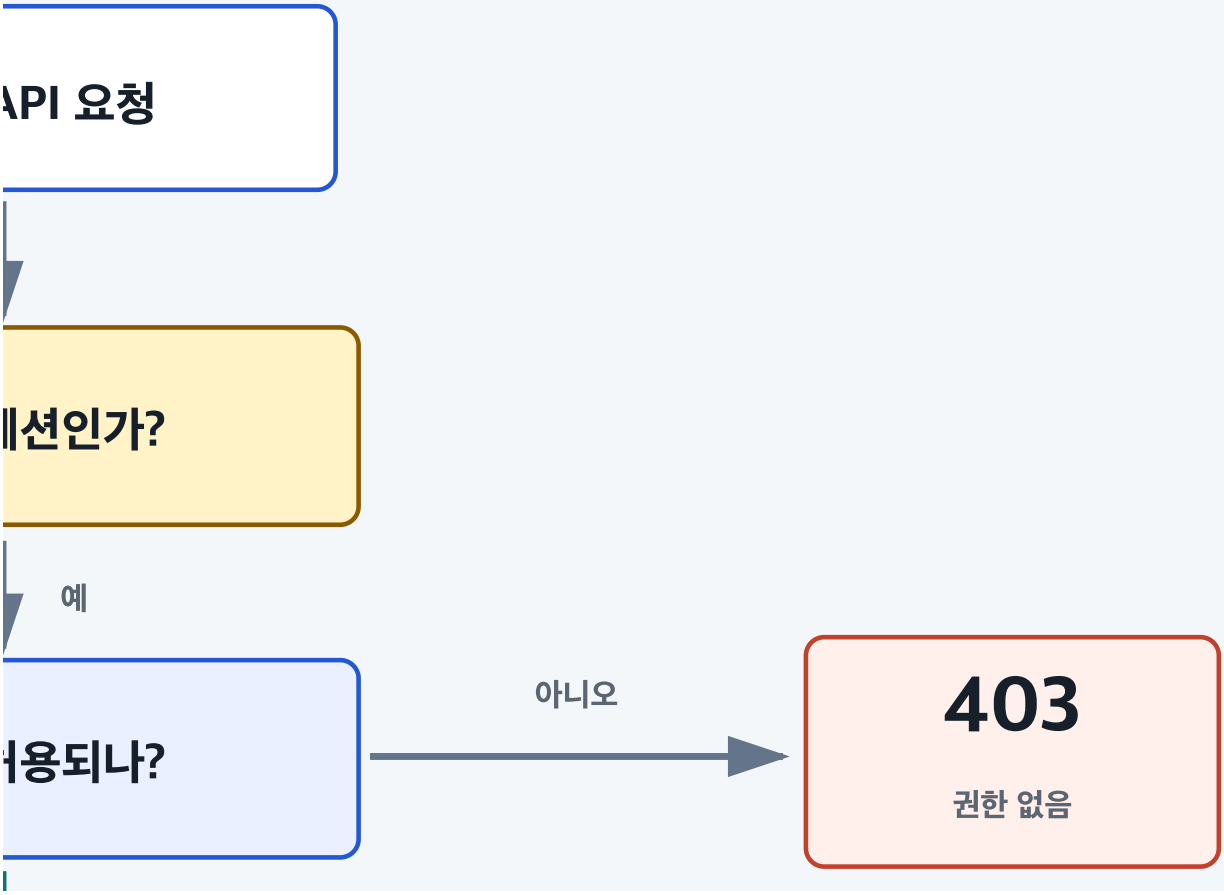


그림 11. 401은 valid identity context가 없으므로 다시 로그인하라는 신호이고, 403은 identity는 알지만 이 action은 허용되지 않았다는 신호입니다.

세션이 없으면 인증 실패, 있지만 행동이 허용되지 않으면 인가 실패다.




API 요청
세션인가?
예
허용되나?
아니오
403
권한 없음

장면	status	stable code	UI 다음 행동	DB 변화
익명 목록 요청	401	`AUTHENTICATION_REQUIRED`	login view	0
만료 session 목록 요청	401	`AUTHENTICATION_REQUIRED`	만료 안내 + login	0
viewer 작성	403	`PERMISSION_DENIED`	조회 전용 설명	0
planner CSRF 누락	403	`CSRF_FAILED`	보안 실패·재시도 제한	0
planner valid 작성	201	-	완료 안내 + 목록 추가	+1

13.1. 403이 로그인 page로 보내지 않는 이유

viewer는 이미 정상 로그인되었습니다. 다시 로그인해도 role이 바뀌지 않으면 결과는 같습니다. UI는 이 계정이 조회 전용임을 설명하고, 필요하면 권한 요청 절차를 안내해야 합니다.

14. Role-based UI로 가능한 행동과 이유를 보여 줍니다

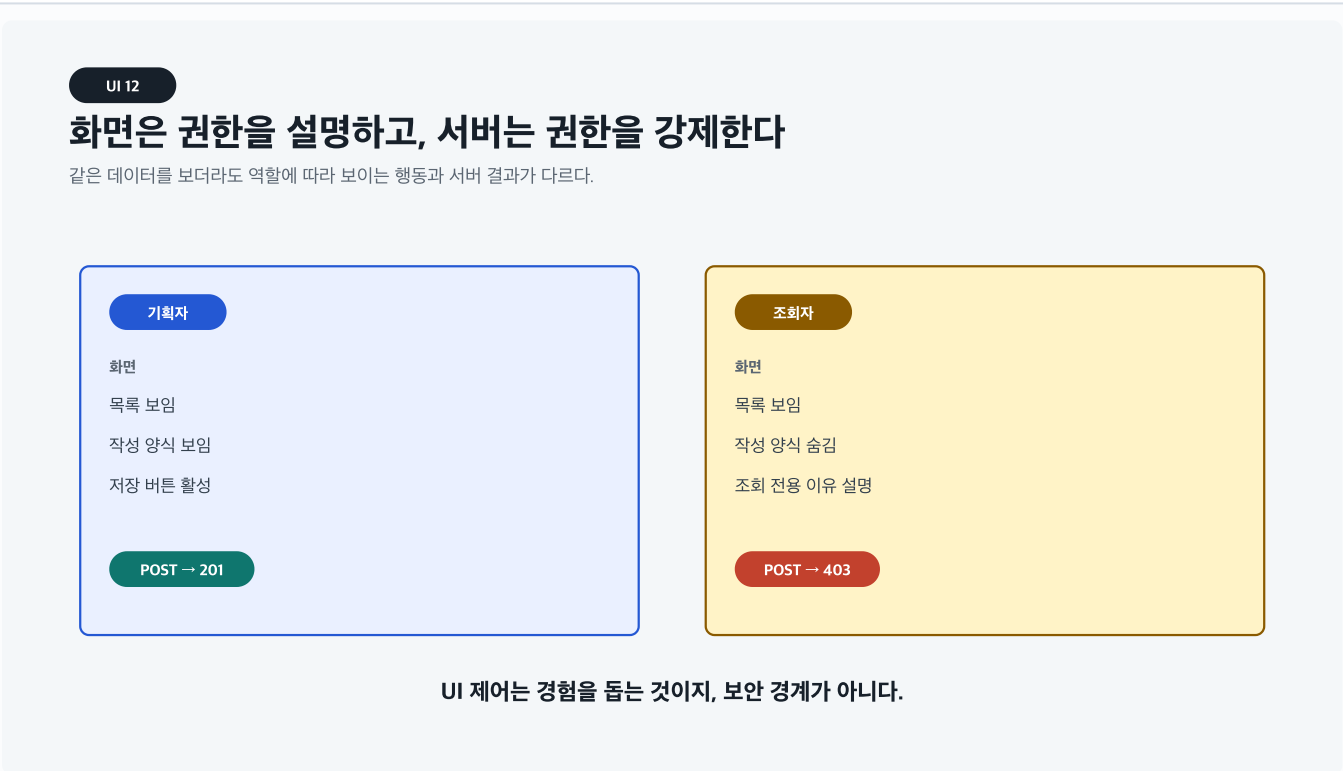


그림 12. 기획자는 작성 양식을 보고, 조회자는 양식 대신 조회 전용인 이유를 보지만, server는 두 요청을 독립적으로 다시 검증합니다.

같은 데이터를 보더라도 역할에 따라 보이는 행동과 서버 결과가 다르다.

기획자

화면

목록 보임

작성 양식 보임

저장 버튼 활성

POST → 201

조회자

화면

목록 보임

작성 양식 숨김

조회 전용 이유 설명

POST → 403

14.1. 기획자 완성 화면

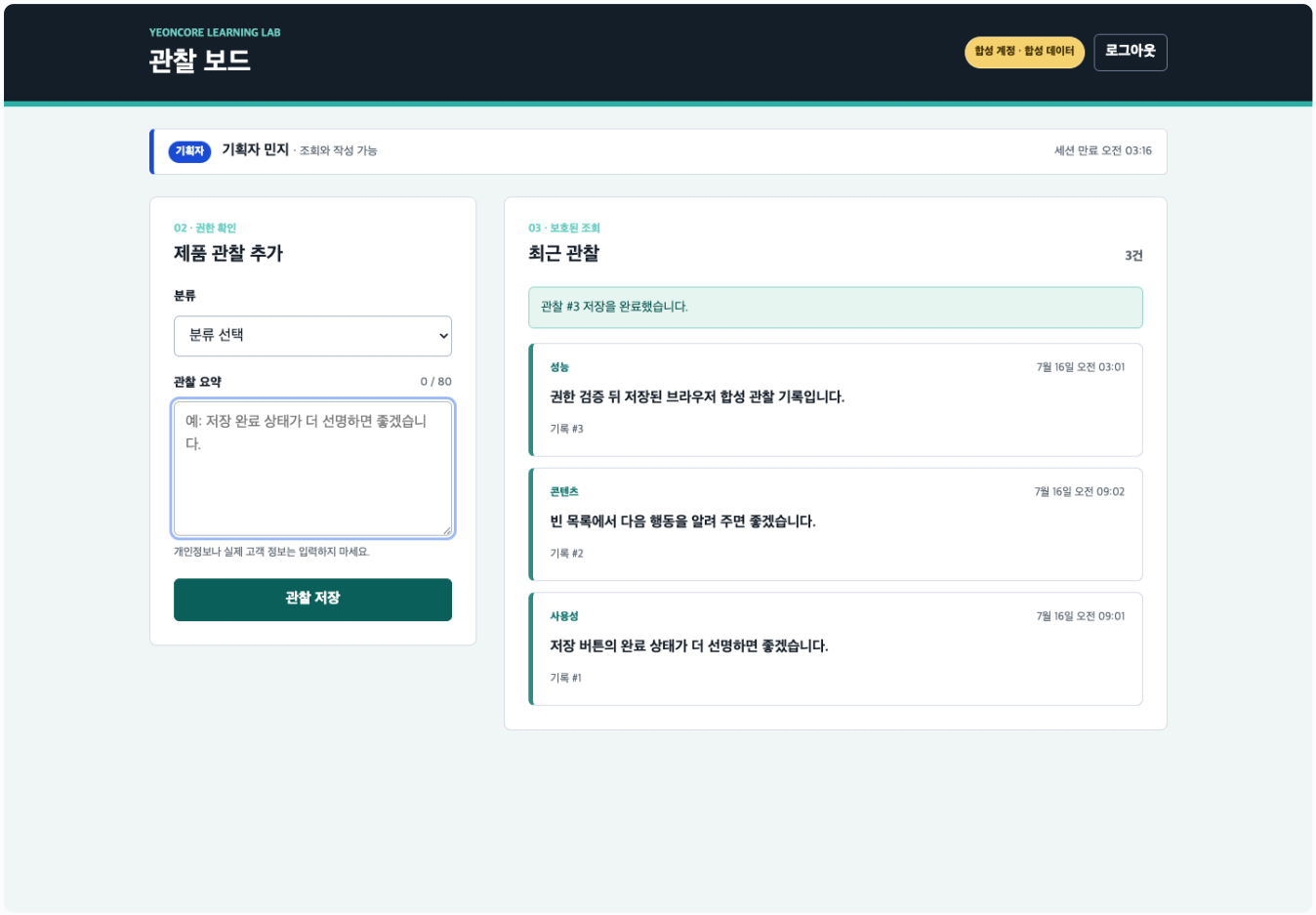


그림 13. 기획자 session은 role chip·만료 시간·작성 양식을 보고, POST 201 후 목록이 2건에서 3건으로 변했습니다.

14.2. 조회자 완성 화면

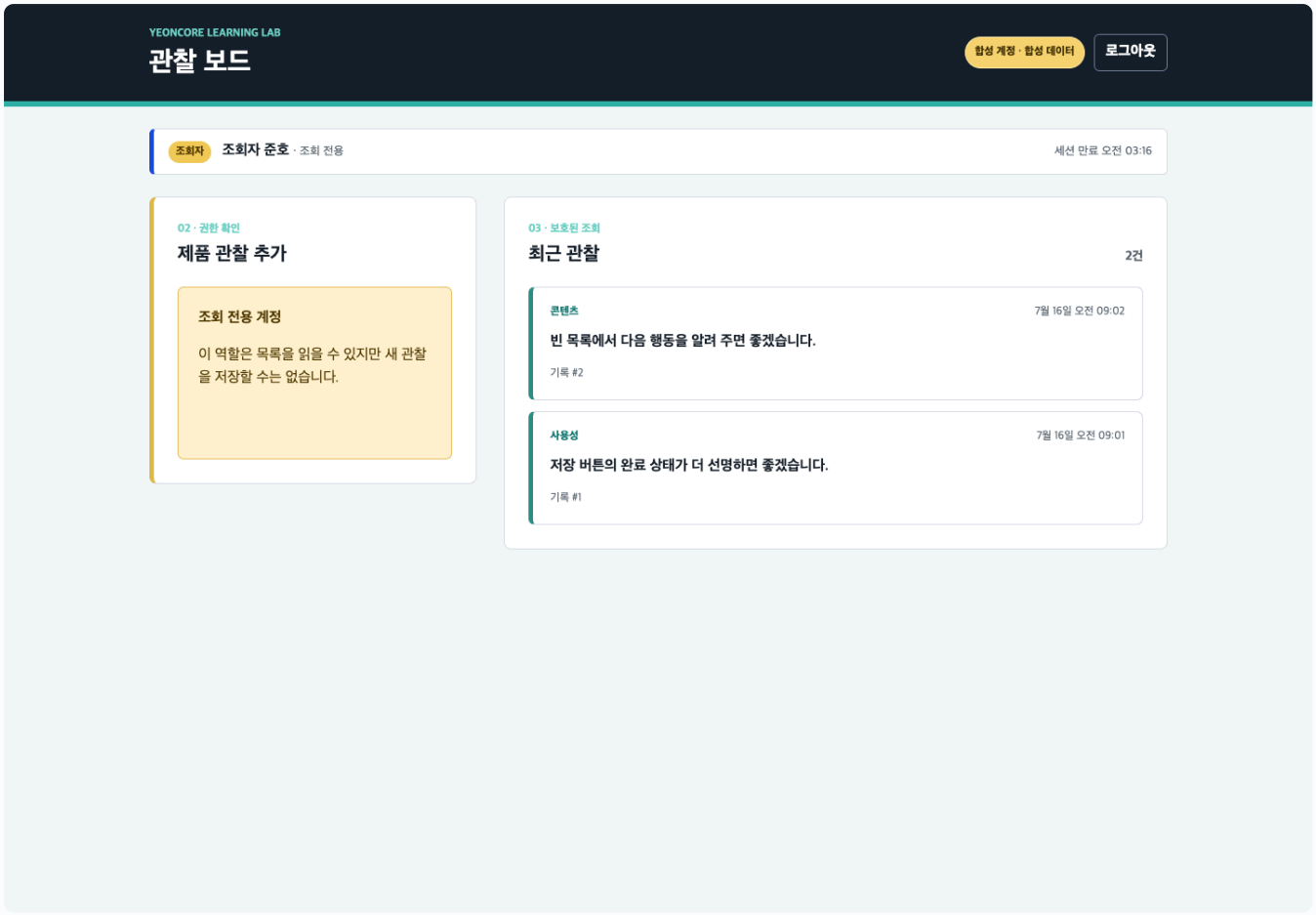


그림 14. 조회자는 보호된 목록 2건을 읽지만, 작성 양식 대신 조회 전용 이유를 보여 줍니다.

14.3. 화면에 보여야 할 세션 정보

표시	이유	표시하지 않을 것
display name	누구로 작업 중인지 확인	password·hash
role label	가능한 행동 예상	내부 permission 전체
session expiry	예상치 못한 만료 완화	raw cookie token
logout command	사용자가 세션 종료	CSRF token 문자열

15. API 계약을 역할별로 확인합니다

method	path	세션	CSRF	permission	성공	주요 실패
GET	`/api/health`	없음	-	-	200	-
GET	`/api/session`	선택	-	-	200 data/session 또는 null	-
POST	`/api/session`	없음	login boundary	-	201 + cookie	401 invalid credentials
DELETE	`/api/session`	필수	필수	authenticated	200 + revoke	401·40 403
GET	`/api/observations`	필수	-	`observation:read`	200	401·40 403
GET	`/api/observations/{id}`	필수	-	`observation:read`	200	401·40 403·40 404
POST	`/api/observations`	필수	필수	`observation:create`	201 + Location	401·40 403·40 4xx·40 5xx

15.1. 오류 code는 status보다 더 정밀한 UI 분기입니다

```
if (error.status === 401) {
  showLogin("세션이 종료되었습니다. 다시 로그인하세요.");
}
```

403에는 `PERMISSION_DENIED`와 `CSRF_FAILED`가 모두 있을 수 있습니다. UI는 stable code를 사용해 권한 부족과 요청 무결성 실패를 다르게 취급할 수 있습니다.

16. User·Session·Observation Table을 관계로 연결합니다

16.1. 핵심 schema

```
CREATE TABLE users (
  id INTEGER PRIMARY KEY,
  username TEXT NOT NULL UNIQUE,
  role TEXT NOT NULL CHECK (role IN ('planner', 'viewer')),
  password_salt BLOB NOT NULL,
  password_hash BLOB NOT NULL,
  password_iterations INTEGER NOT NULL,
  active INTEGER NOT NULL DEFAULT 1
);

CREATE TABLE sessions (
  id INTEGER PRIMARY KEY,
  token_hash BLOB NOT NULL UNIQUE,
  user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  csrf_token TEXT NOT NULL,
  created_at TEXT NOT NULL,
  expires_at TEXT NOT NULL
);
```

16.2. Schema 불변 규칙

규칙	막는 실패
unique username	같은 login id 중복
role CHECK	정책에 없는 role 저장
salt length >= 16	짧거나 빈 salt
hash length = 32	다른 형식의 잘못된 digest
iteration >= 1000	잘못된 0례 cost
token hash unique	세션 key 충돌
session user FK	존재하지 않는 user의 session

DB constraint는 server permission 정책을 대체하지 않습니다. DB는 저장 형식과 관계의 마지막 무결성 경계입니다.

17. Browser State와 Token Storage를 분리합니다

```
const state = {
  session: null,       // user, role, csrfToken, expiresAt
  observations: [],
  view: "loading",
};
```

17.1. 어디에 무엇을 둔는가

data	위치	page refresh	JavaScript 읽기	이유
raw session token	HttpOnly cookie	유지	불가	browser가 자동 전송
CSRF token	JavaScript memory	사라짐	가능	GET session으로 다시 받음
user·role·expiry	JavaScript memory	사라짐	가능	UI state 렌더
password	form control 일시	사라짐	제출 중만	login 후 reset

실습은 `localStorage` · `sessionStorage`에 session token을 저장하지 않습니다. 그러나 HttpOnly cookie가 XSS의 모든 피해를 막는 것은 아닙니다. 안전한 DOM API, CSP, input·output 처리가 함께 필요합니다.

17.2. 안전한 동적 표시

```
userName.textContent = state.session.user.displayName;
roleChip.textContent = state.session.user.role === "planner" ? "기획자" : "조회자";
```

사용자·API data를 `innerHTML`로 파싱하지 않고 text node로 넣습니다.

18. Security는 Negative Test Journey로 증명합니다

그림 15. 정상 login 한 번은 보호 증거가 아닙니다. 없는 자격, 권한 부족, CSRF 누락, logout 폐기가 각각 기대한 지점에서 멈춰야 합니다.

정상 로그인 하나만으로는 세션, CSRF, 권한, 폐기가 작동하는지 알 수 없다.

증거

HTTP 상태 · 오류 코드 · 쿠키 속성

기획자

POST – CSRF

기획자

POST + CSRF

로그아웃 후

GET

묶음

· DB 행 변화 · 로그아웃 후 재시도

18.1. 요구하는 테스트 증거

#	starting state	request	expected	보호하는 것
1	cookie 없음	GET list	401	anonymous access
2	없는 username	POST login	generic 401	account enumeration
3	존재 user·틀린 password	POST login	같은 generic 401	credential detail
4	viewer session	GET list	200	read permission
5	viewer session + valid CSRF	POST create	403	write permission
6	planner session + no CSRF	POST create	403	cross-site state change
7	planner session + valid CSRF	POST create	201	intended write
8	revoked session cookie	GET list	401	logout reuse

18.2. DB 불변 증거

401·40 403 요청 전후의 observation row count가 같아야 합니다. status만 보고 끝내지 않고 forbidden request가 side effect를 남기지 않았는지 확인합니다.

19. 생성기가 남긴 Evidence Packet을 읽습니다

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

evidence	판정 질문	기대 핵심
reset	시작 상태가 같은가	users 2·rows 2·sessions 0
tests	계약·DB·policy·browser 코드가 닫혔는가	23 tests OK

evidence	판정 질문	기대 핵심
audit	소스·저장 구조의 guardrail이 있는가	16/16 PASS
probe	역할별 HTTP 여정이 실제로 맞는가	401·40 403·40 201·40 logout 401
auth storage	raw token이 없고 cost·salt가 있는가	hash32·salt16·600k·raw false
versions	어떤 runtime에서 검증했는가	Python·SQLite·executable

19.1. 실제 검증 결과

```
Python 3.12.13: 23 tests PASS, 16/16 audit PASS, contract probe PASS
Python 3.14.5 : 23 tests PASS, 16/16 audit PASS, contract probe PASS
Desktop Chrome: viewer form hidden, planner POST 201, count 2→3
Mobile Chrome : planner form visible, horizontal overflow 0
Console errors: 0
Failed requests: 0
```

20. 75분 실습 Mission

Mission A · 실습 묶음 생성 · 10분

```
./02_Labs/G08_Fullstack/L08-02_create-auth-permission-practice.sh
```

합격:

```
CHECKS: PASS
evidence 파일 6개
실습 target이 이미 있을 때 exit 2
```

Mission B · 계정별 화면 비교 · 15분

1. viewer로 login하고 조회 전용 안내를 기록합니다.
2. logout합니다.
3. planner로 login하고 작성 양식을 기록합니다.

4. 두 화면의 같은 점·다른 점을 표로 적습니다.

Mission C · DevTools에서 세 경계 찾기 · 15분

request	확인할 증거
POST session	201·Set-Cookie flags·JSON에 raw token 없음
GET observations	Cookie는 browser가 보냄·200
POST observations	X-CSRF-Token·201·Location
DELETE session	X-CSRF-Token·200·Max-Age=0

Mission D · Negative 여정 · 20분

`scripts/contract_probe.py`를 실행하고 다음을 자신의 말로 설명합니다.

```
왜 anonymous는 401인가 =
왜 viewer read는 200인가 =
왜 viewer create는 403인가 =
왜 planner no-CSRF는 403인가 =
왜 planner valid create는 201인가 =
왜 logout 후는 401인가 =
```

Mission E · Reset·재현 · 10분

```
python3 scripts/reset_db.py
python3 -m unittest discover -s tests -v
python3 scripts/audit_auth.py
python3 scripts/contract_probe.py
```

Mission F · 한계 문장 · 5분

```
이 실습이 증명한 것 =
이 실습이 증명하지 않은 것 =
운영 전환 전에 필요한 것 =
```

21. 자주 실패하는 패턴

패턴	보이는 증상	숨은 문제	수정
role을 browser state만 보고 판단	button은 숨겨짐	direct API 요청은 통과	server policy check
password를 암호화해 복호화	login은 됨	key 유출 시 원문 노출	password KDF로 검증
모든 계정에 같은 salt	hash는 다르게 보임	같은 password 식별 쉬움	unique random salt
session token을 DB에 원문 저장	조회 편리	DB 유출이 session hijack으로 즉시 연결	token hash 저장
session token을 localStorage에 저장	page refresh 편리	injected script가 읽기 쉬움	HttpOnly cookie
SameSite만 설정	일부 CSRF 감소	단독 방어 과신	CSRF token + framework 방어
logout에서 cookie만 삭제	현재 browser는 logout	훔친 token은 재사용	server session revoke
401·40 403을 모두 login으로 전환	user가 반복 login	permission은 바뀌지 않음	401·40 403 UX 분리
정상 시나리오만 테스트	demo는 성공	보호 경계 미검증	negative matrix
로컬 HTTP를 운영 모델로 간주	localhost에서는 작동	네트워크 노출	HTTPS·Secure·mature stack

22. 이 Local Lab의 Security 한계를 명확히 적습니다

포함된 방어

- unique salt + PBKDF2-HMAC-SHA256 600,000 iterations
- generic login failure + dummy KDF work
- opaque session token + DB token hash
- HttpOnly·SameSite=Lax·Path·Max-Age cookie
- server expiry·login rotation·logout revocation
- synchronizer CSRF token for POST create and DELETE logout

- deny-by-default role-action policy
- protected request마다 session·permission 재검증
- parameterized SQL·transaction·schema constraint
- textContent·CSP·no-store response

포함되지 않은 방어

- production TLS termination·HSTS·Secure cookie 실제 적용
- MFA·passkey·password reset·email verification
- login throttling·credential stuffing detection·bot defense
- OAuth·OIDC·SAML·SSO·SCIM
- device·IP·risk-based session policy
- session store encryption·key rotation·distributed revocation
- login CSRF의 mature framework defense
- audit logging·alerting·incident response·account recovery
- privacy notice·retention·legal basis·data subject request
- independent security review·penetration test

과장 금지
23개 테스트와 16개 audit가 통과했다고 해서 이 app이 production-ready거나 완전히 안전하다는 뜻은 아닙니다. 이는 명시한 local learning contract가 재현됨을 증명합니다.

23. 실무 전환 질문

영역	물어야 할 질문
identity	직접 계정을 구현할 이유가 있는가, 검증된 IdP를 쓸 수 있는가
assurance	어떤 행동에 MFA·step-up가 필요한가
session	idle·absolute timeout은 얼마인가, 다중 device를 허용하는가
authorization	역할만으로 충분한가, resource ownership·organization·context가 필요한가
recovery	password reset·account unlock·lost factor recovery를 누가 승인하는가

영역	물어야 할 질문
abuse	brute force·credential stuffing·session theft를 어떻게 탐지하는가
audit	누가 언제 어떤 권한으로 무엇을 했는지 어떤 data 없이 남길것인가
incident	session key·DB·IdP 사고 시 전체 폐기·재인증 절차가 있는가

24. Eight Done Gates로 완료를 판정합니다

그림 16. password·session·CSRF·permission을 따로 보호 계층으로 읽고, 성공 201보다 여러 실패 상태로 완료를 증명합니다.

비밀번호부터 세션, CSRF, 역할 정책, 실패 증거까지 하나의 연결된 체계다.

비밀번호

원문 저장 금지

salt + KDF + cost

세션

임의 토큰 쿠키

회전 + 만료 + 폐기

완료 증거 6

- ✓ 익명 401
- ✓ 조회자 403
- ✓ 기획자 201
- ✓ 로그아웃 200

보호 요청

세션 + CSRF

매 요청 권한 확인

역할 정책

planner / viewer

deny by default

- ✓ **CSRF 403**
- ✓ **재접근 401**

gate	질문	합격 증거
1 Identity	합성 user를 안전한 password verifier로 검증하는가	unique salt·KDF·generic 401
2 Session	raw token을 server hash에 연결하는가	raw DB column 없음·HttpOnly cookie
3 Lifecycle	login·expiry·logout이 session을 바꾸는가	rotation·15분·revoke
4 CSRF	state change가 별도 토큰을 요구하는가	no token 403
5 Permission	모든 보호 request에서 정책을 재검증하는가	viewer read 200·create 403
6 UI	role별 가능 행동과 이유가 보이는가	planner form·viewer read-only panel
7 Negative	401·40 403·40 201·40 logout 401을 재현하는가	probe PASS·23 tests
8 Boundary	local lab과 production 요구사항을 구분하는가	TLS·MFA·throttling·IdP 한계 문장

25. 셀프 테스트 12문항

문제 1

인증, 세션, 인가가 각각 답하는 질문을 한 문장씩 적으세요.

정답 보기
인증은 자격 정보가 알려진 사용자인지, 세션은 그 로그인 맥락이 아직 유효한지, 인가는 그 사용자가 특정 action을 해도 되는지를 묻습니다.

문제 2

viewer가 login한 뒤 관찰 작성을 요청했습니다. 401과 403 중 어느 것이며 이유는 무엇인가요?

정답 보기
403입니다. viewer의 identity와 session은 유효하지만
`observation:create`
permission이 없습니다.

문제 3

없는 username과 틀린 password에 같은 login 오류를 사용하는 이유는 무엇인가요?

정답 보기
응답만으로 계정이 실제로 존재하는지 탐색하는 account enumeration을 줄이기 위해서입니다.

문제 4

password에 unique salt와 KDF cost를 사용하는 이유를 각각 적으세요.

정답 보기
unique salt는 같은 password도 계정마다 다른 hash를 만듭니다. KDF cost는 password 후보 1회를 시도하는 비용을 높입니다.

문제 5

DB session table에 raw token 대신 token hash를 두는 이유는 무엇인가요?

정답 보기
session DB snapshot이 노출되었을 때 공격자가 DB의 값을 cookie로 바로 재사용하는 위험을 줄이기 위해서입니다.

문제 6

HttpOnly와 Secure가 각각 제한하는 것은 무엇인가요?

정답 보기
HttpOnly는 JavaScript의 cookie 읽기를 제한하고, Secure는 cookie를 HTTPS connection으로만 전송하게 합니다.

문제 7

Browser `Max-Age=900`만 있으면 server `expires_at`이 필요 없는가요?

정답 보기
필요합니다. server는 browser cookie 상태를 신뢰하지 않고 매 요청에서 자신의 absolute expiry를 확인해야 합니다.

문제 8

CSRF token은 어디에 저장되고 어떻게 다시 server로 가나요?

정답 보기

server session row에 저장되고 login·session response로 JavaScript memory에 전달됩니다. 상태 변경 요청의 `X-CSRF-Token` header로 다시 보냅니다.

문제 9

Protected POST에서 session, CSRF, permission, input, DB 순서를 쓰세요.

정답 보기

session 확인 → CSRF 일치 → role-action permission 확인 → JSON·업무 input 검증 → DB transaction → 201 response 순입니다.

문제 10

Logout에서 browser cookie와 server session 중 어느 쪽을 삭제해야 하나요?

정답 보기

두 쪽 모두입니다. server session row를 revoke하고 browser에 Max-Age=0 cookie를 보냅니다.

문제 11

화면에서 viewer의 작성 양식을 숨기면 server authorization을 생략해도 되나요?

정답 보기

안 됩니다. UI는 사용 경험을 돕지만 변조 가능합니다. server가 매 protected request에서 permission을 재검증해야 합니다.

문제 12

이 lab을 production-ready로 불러서는 안 되는 한계를 네 개 적으세요.

정답 보기

예시: local HTTP라 TLS·Secure cookie를 실제 적용하지 않았고, MFA·password reset·login throttling·mature login CSRF·SSO·distributed session·audit logging·security review가 없습니다. 이 중 네 개를 적으면 됩니다.

26. 현장 적용 Checklist

Identity·Password

- [ ] 직접 identity를 구현할 이유와 mature provider 대안을 비교했다.
- [ ] password를 원문·복호화 가능 형식으로 저장하지 않는다.
- [ ] 검증된 password KDF·unique salt·적절한 cost를 사용한다.
- [ ] 없는 계정과 틀린 password의 외부 응답을 통일한다.
- [ ] throttling·MFA·recovery·monitoring을 설계했다.

Session·Cookie

- [ ] session token은 CSPRNG로 충분히 길게 생성한다.
- [ ] DB에 raw token을 저장하지 않는다.
- [ ] login·privilege change에 session rotation을 적용한다.
- [ ] idle·absolute expiry와 logout revocation을 server가 확인한다.
- [ ] cookie에 HttpOnly·Secure·SameSite·Path·적절한 expiry를 설정한다.

CSRF·Authorization

- [ ] 상태 변경 request가 검증된 CSRF defense를 거친다.
- [ ] 매 request에서 server가 user·action·resource를 재검증한다.
- [ ] 정책에 없는 role·action은 기본 거부한다.
- [ ] 401과 403을 회복 행동에 맞게 구분한다.
- [ ] UI 제어를 security boundary로 간주하지 않는다.

Evidence·Operations

- [ ] anonymous·invalid login·wrong role·missing CSRF·expiry·logout 테스트가 있다.
- [ ] 거부된 request가 DB side effect를 남기지 않는다.
- [ ] version·reset·test·storage snapshot·HTTP proof를 남긴다.
- [ ] audit log에 secret·password·raw token을 남기지 않는다.
- [ ] incident 시 session 전체 폐기·재인증·회복 절차가 있다.

27. 공식 출처와 적용 원칙

출처	이 교재에 적용한 원칙
OWASP Authentication Cheat Sheet	generic login responses, TLS, MFA·throttling 운영 경계
OWASP Password Storage Cheat Sheet	Argon2id 우선, PBKDF2-HMAC-SHA256 600k 학습 선택, salt·cost
OWASP Session Management Cheat Sheet	추측 어려운 session id, cookie flags, rotation·expiry·logout
OWASP Authorization Cheat Sheet	least privilege, deny by default, every request 권한 검증
OWASP CSRF Prevention Cheat Sheet	synchronizer token, SameSite를 defense-in-depth로 사용
Python 3.12 hashlib	`pbkdf2_hmac`, iteration·salt 기능 판독
Python 3.12 secrets	password·session·CSRF 용 암호학적 random token 생성
MDN Set-Cookie	HttpOnly·Secure·SameSite·Path·Max-Age 속성 의미

출처를 읽는 방법

1. Cheat sheet를 완성 코드 복사본으로 사용하지 않습니다.
2. 우리 threat model·identity lifecycle·regulation·framework에 맞게 적용합니다.
3. 이 교재의 local implementation과 production recommendation을 분리해 기록합니다.
4. 보안 판정은 신규 정보·framework version·운영 환경으로 재검토합니다.

다음 매뉴얼: M08-03 실제 서비스처럼 오류와 상태 처리하기