강동현 · Backend Engineer

같은 조회를 query 수, 읽은 row, buffer, 실행 계획으로 나눠 비교했습니다

> 멘토링 과제로 진행한 Feed Query 성능 비교입니다. N+1, Batch Fetch, Projection, Top-N, index 의존성, pagination을 같은 데이터셋과 같은 측정 조건에서 비교했습니다. 뒤쪽에는 TechLog Backend에서 게시 상태와 module 경계, 인증 경계를 정리한 내용이 이어집니다.

2,022 → 23
N=1,000 요청 1건의 PreparedStatement

1,569 → 0
Projection 적용 후 entity load

2,000 → 20
100번째 페이지에서 읽은 row

PostgreSQL 16 · Hibernate Statistics · EXPLAIN (ANALYZE, BUFFERS) · Testcontainers · Java 21 · Spring Boot

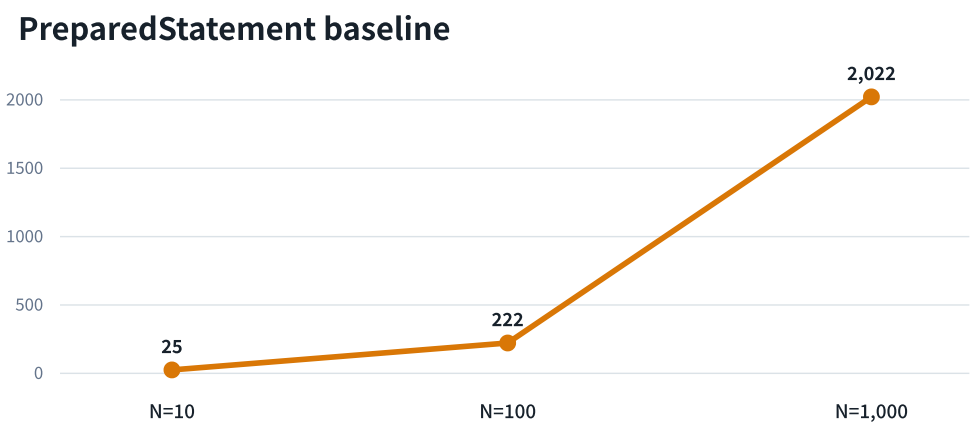
GitHub	**github.com/donghyeon-ka**
TechLog	hyeonworks.com
Email	donghyeonka68@gmail.com

01 · Performance Lab · N+1 Baseline

요청 1건이 SQL 2,022개로 늘어났습니다

목록 1페이지를 여는 데 실제로 몇 번 질의하는가? · 별도 멘토링 과제 · Feed Query Performance Lab

`loadFeed(0, N)` 한 번이 N=1,000에서 PreparedStatement 2,022개를 만들었습니다. content 1 + count 1 + User 20 + Page 1,000 + Highlight 1,000으로 분해됩니다. 부모 행이 늘어난 만큼 질의가 따라 늘어나는 구조였습니다.



N=1,000 decomposition

content	1
count	1
User ToOne	20
Page ToOne	1,000
Highlight collection	1,000

N	Highlight collection	PreparedStatement	median (5)	max (5)
10	10	25	32.8 ms	36.1 ms
100	100	222	85.9 ms	108.3 ms
1,000	1,000	**2,022**	193.7 ms	238.4 ms

PostgreSQL 16 Testcontainers · Hibernate Statistics · 동일 JVM · warm DB cache · single thread. 지연은 7회 중 앞 2회를 warm-up으로 제외한 5회의 median / max입니다.

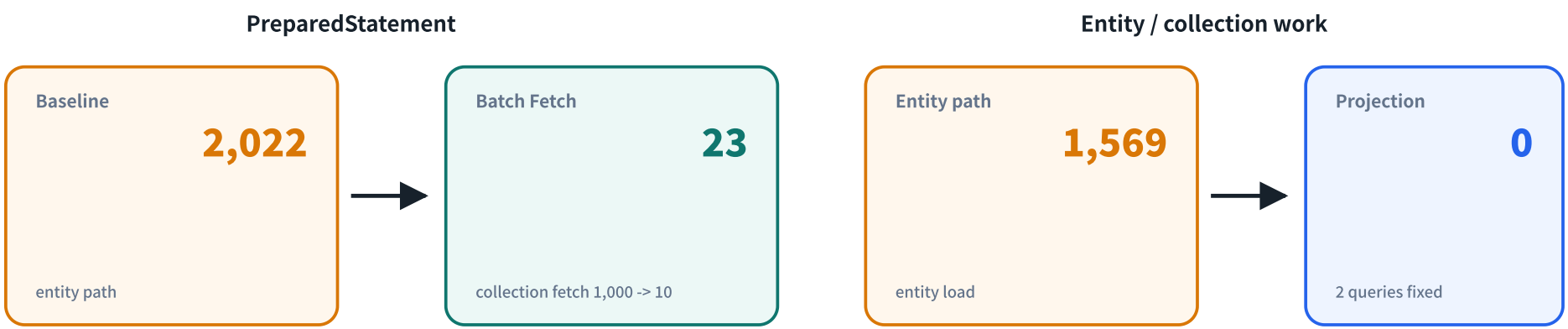
02 · Performance Lab · Batch / Projection

PreparedStatement 2,022 → 23, entity load 1,569 → 0

왕복을 줄이는 것과 읽기 경로를 분리하는 것은 같은 선택인가? · 별도 멘토링 과제 · Feed Query Performance Lab

같은 `loadFeed(0, N)`에서 조회 전략만 바꿔 N=1,000으로 다시 측정했습니다.

N=1,000 strategy comparison



Batch Fetch는 PreparedStatement를 2,022 → 23으로, collection fetch를 1,000 → 10으로 줄였습니다. Projection은 query를 2개로 고정하고 entity load를 1,569 → 0으로 없앴습니다.

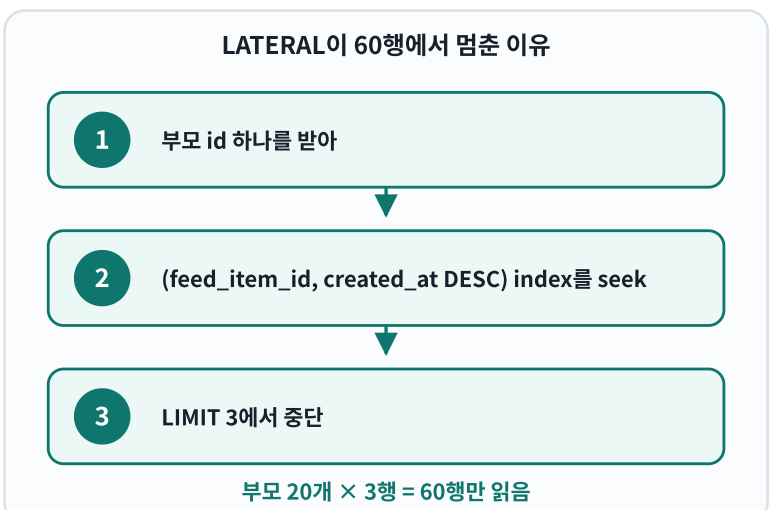
Batch는 기존 entity 조회 구조를 그대로 두고 왕복만 줄인 최적화이고, Projection은 읽기 전용 contract를 따로 만들어 경로 자체를 나눈 선택입니다. Projection 쪽은 수정 경로에서 쓰던 entity 기능을 쓸 수 없으므로, 목록처럼 읽기만 하는 화면에만 적용했습니다.

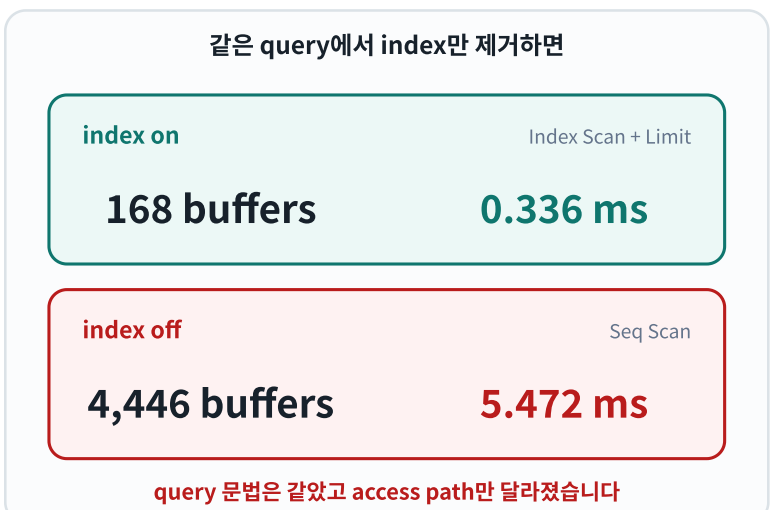
03 · Performance Lab · Top-N & Index

LATERAL은 index가 있을 때만 60행에서 멈췄습니다

부모별 최신 3건을 가져오려고 전체 row를 정렬해야 하는가? · 별도 멘토링 과제 · Feed Query Performance Lab

부모 20개에서 각각 최신 3건을 가져오는 질의를 Window Function / LATERAL / 2-step으로 비교한 뒤, 이긴 계획이 무엇에 기대고 있는지 다시 확인했습니다.





- Window: 60 rows · 430 buffers · 1.552 ms
- LATERAL: 60 rows · **204 buffers · 0.323 ms**
- 2-step: 1,509 rows · 430 buffers · 1.686 ms

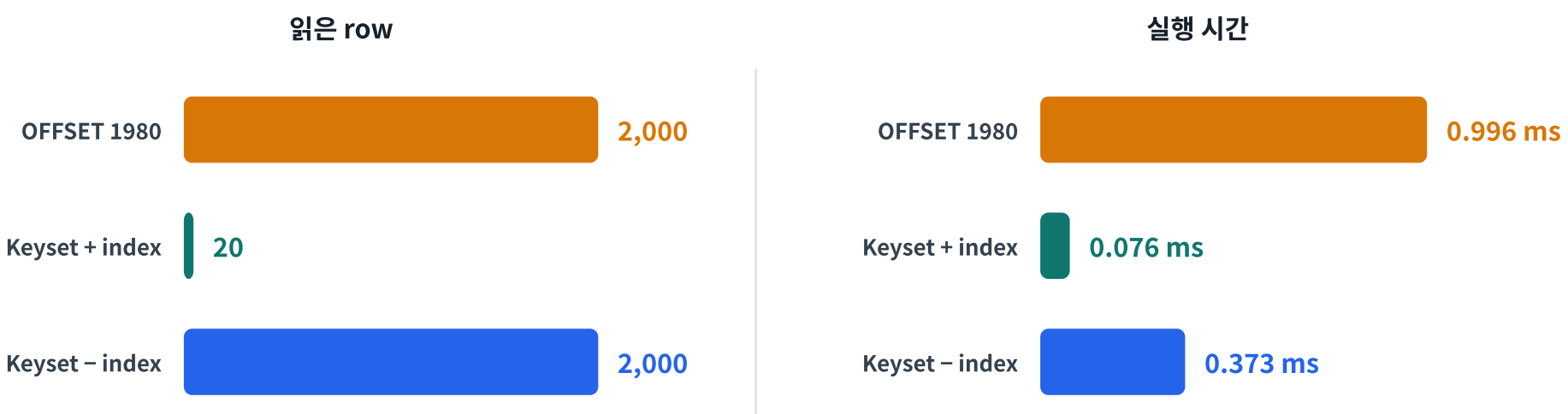
LATERAL이 빠른 기법이어서가 아니라, 부모별로 원하는 row를 바로 찾을 composite index가 있어서 `LIMIT 3`에서 멈출 수 있었습니다. 작은 K와 큰 group, 그리고 해당 index가 함께 있는 조건에서 측정한 결과입니다.

04 · Performance Lab · Pagination

100번째 페이지에서 OFFSET은 2,000행, Keyset은 20행을 읽었습니다

깊은 페이지에서 OFFSET은 무엇을 더 하고 있는가? · 별도 멘토링 과제 · Feed Query Performance Lab

seed 2,000건 · page size 20 · 같은 정렬 조건에서 마지막 페이지를 요청했습니다. OFFSET은 건너뛸 행도 일단 읽은 뒤 버리기 때문에 페이지가 깊어질수록 읽는 양이 같이 늘어납니다.



buffers — OFFSET 141 · Keyset + index 1 · Keyset − index 141

- OFFSET 1980: 2,000 rows · 141 buffers · 0.996 ms
- Keyset + index: **20 rows · 1 buffer · 0.076 ms**

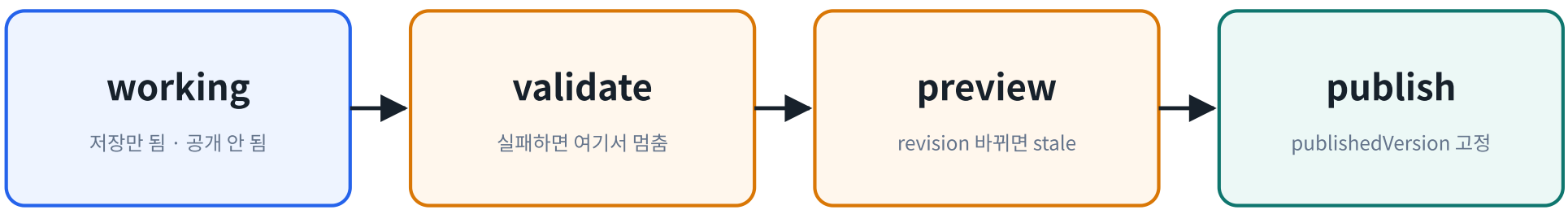
Keyset이 항상 빠른 것은 아니고, 정렬 key가 안정적이고 cursor로 이어서 탐색하는 화면에서만 성립합니다. 같은 keyset 조건이어도 정렬 index를 제거하면 다시 전체 scan으로 돌아갔습니다.

05 · TechLog · Publication Workflow

저장되는 시점과 공개되는 시점을 분리했습니다

Studio에서 고치는 중인 글이 그대로 공개되어도 되는가? · TechLog Backend · 개인 프로젝트

Studio에서는 글을 고치는 동안 저장이 계속 일어납니다. 초기 구현에서는 저장된 내용이 곧 공개 화면의 내용이었기 때문에, 검증을 통과하지 못한 문서도 링크만 알면 보였습니다.



Public read는 publication status와 publishedVersion만 봅니다

working version은 공개 화면에서 조회되지 않습니다

저장되는 시점과 공개되는 시점이 다른 version을 가리킵니다

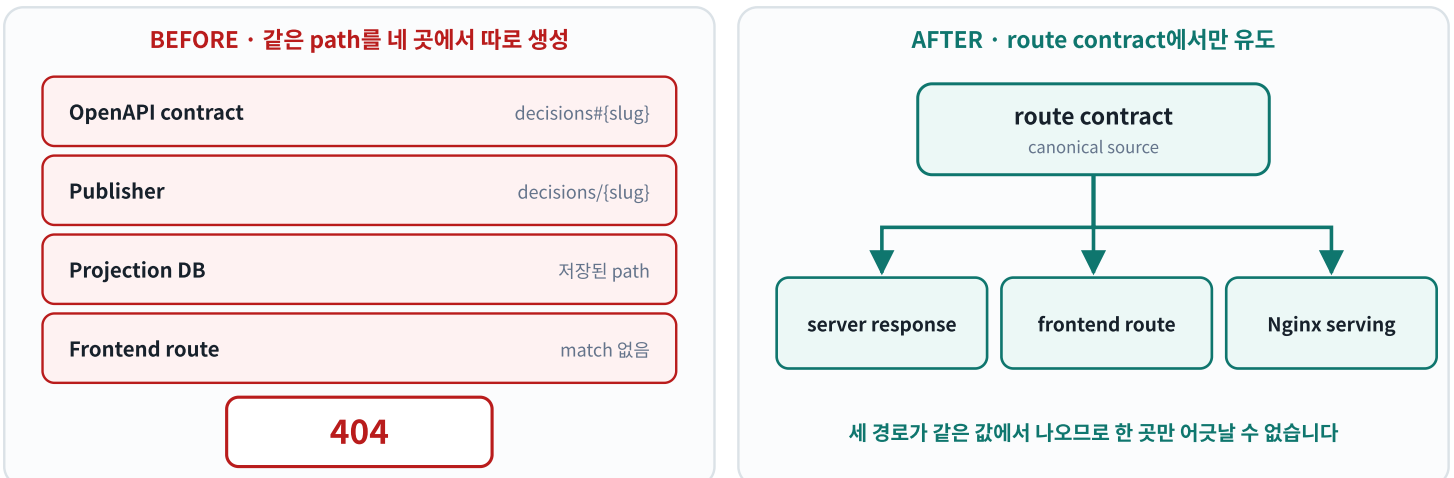
Create / Save / Validate / Preview / Publish를 각각 별도 use case로 나누고 working version과 publishedVersion을 따로 저장합니다. 검증에 실패하면 publish 단계로 넘어가지 않고, 문서나 dependency의 revision이 바뀌면 이전 preview는 stale로 표시됩니다. 공개 화면은 endpoint 여러 개의 응답을 조합해 게시 여부를 추측하지 않고 publication status와 publishedVersion만 봅니다.

06 · TechLog · Contract Drift

게시까지 끝난 글의 링크가 404로 끝났습니다

같은 public path를 몇 군데에서 만들고 있었는가? · TechLog Backend · 개인 프로젝트

Project Decision은 publish까지 끝났고 목록에도 보였지만, 링크를 누르면 404가 났습니다. contract, publisher, projection, frontend route가 각각 같은 public path를 따로 만들고 있었고, 그중 한 곳만 규칙이 달라도 목록과 상세가 어긋났습니다.



지금은 route contract 한 곳에서 path를 만들고, 경계를 넘을 때마다 같은 정의를 쓰는지 확인합니다.

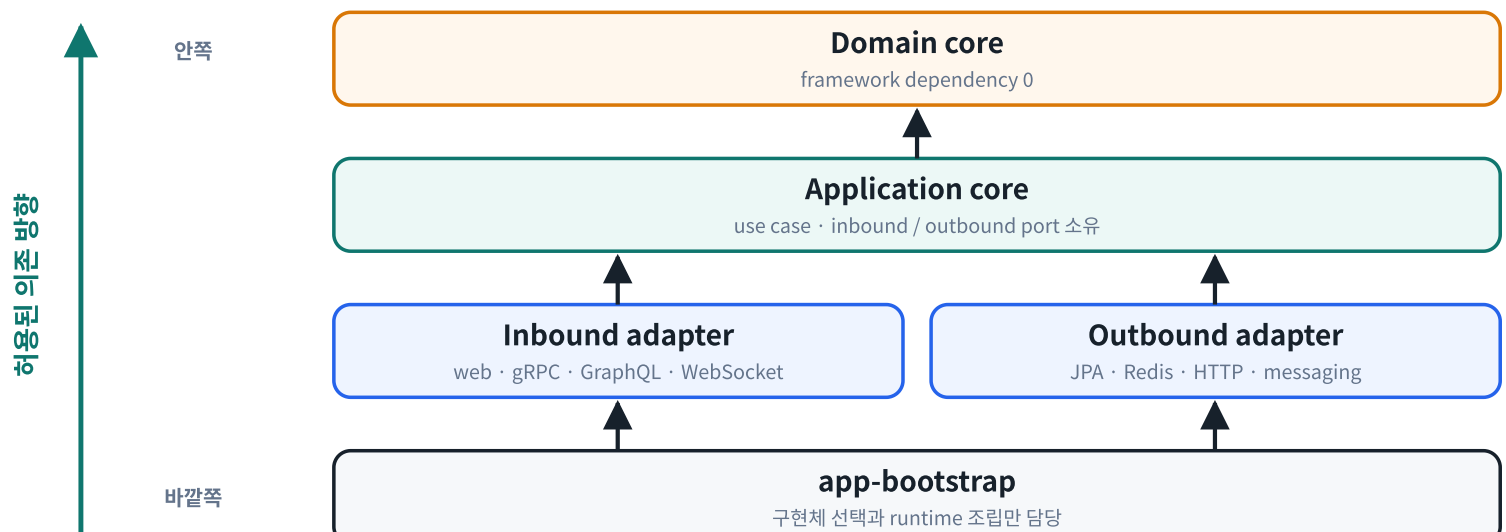
- OpenAPI operation과 RecordKind가 adapter 등록에서 빠졌는지 검사
- server가 응답한 path가 frontend route로 실제 해석되는지 검사
- Nginx serving pattern도 같은 route contract에서 유도
- RecordKind는 `Record<RecordKind, _>`와 exhaustive switch로 누락이 컴파일 단계에서 드러나게 함

07 · TechLog · Dependency Direction

의존 방향을 사람 기억이 아니라 build에서 막습니다

module 18개에서 참조해도 되는 방향을 누가 기억하는가? · TechLog Backend · 개인 프로젝트

web / gRPC / GraphQL / WebSocket과 outbound adapter가 늘면서, 어떤 module이 어떤 module을 참조해도 되는지가 각 Gradle 파일에 흩어졌습니다. 규칙을 리뷰에서 걸러내는 방식으로는 module 18개를 감당하기 어려웠습니다. 허용 edge를 `modules.json`에 한 번 선언하고 실제 project dependency와 비교합니다.



app-bootstrap은 구현체를 고르지만 application contract를 소유하지 않습니다

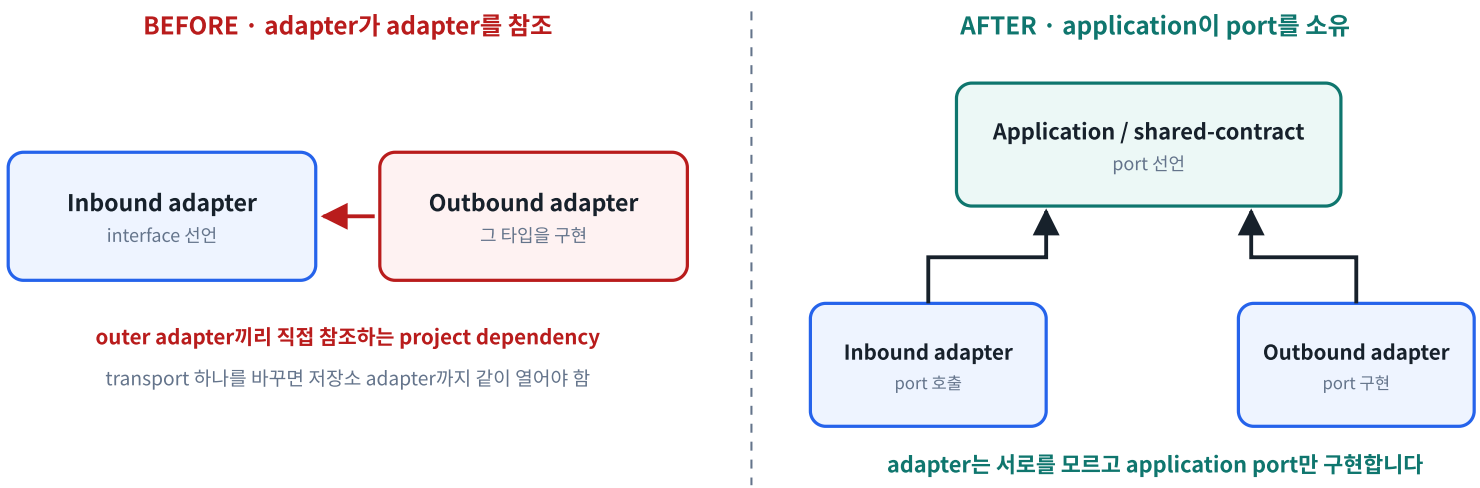
- `domain-core` / `shared-contract`는 허용 project dependency 0
- 선언하지 않은 project edge는 Gradle `check`에서 실패
- domain의 framework dependency와 application의 adapter / transport dependency는 ArchUnit class graph에서 실패

08 · TechLog · Contract Ownership

inbound adapter가 만든 interface를 outbound adapter가 구현하고 있었습니다

adapter 하나를 바꿀 때 어디까지 같이 고쳐야 하는가? · TechLog Backend · 개인 프로젝트

inbound adapter가 자기 interface를 정의하고 outbound adapter가 그 타입을 구현하면서, outer adapter끼리 직접 참조하는 project dependency가 생겼습니다. transport 하나를 바꾸면 관계없는 저장소 adapter까지 같이 열어야 했습니다.



port를 application이 소유하도록 옮겼습니다. 이제 adapter는 서로를 모르고 application port만 구현합니다.

- application이 호출하는 동작은 application port로 선언
- 여러 adapter가 같이 봐야 하는 최소 타입만 `shared-contract`에 둠
- `app-bootstrap`은 구현체 선택과 runtime 조립만 담당

09 · Authentication · 패턴 비교

네 가지 인증 구조를 같은 네 가지 질문으로 비교했습니다

구조 이름 말고 무엇을 기준으로 고를 것인가? · 4-pattern 인증 실험 · Authorization Code + PKCE

SPA / Token Mediator / BFF / Forward-Auth를 같은 Authorization Code + PKCE 흐름으로 각각 구현했습니다. 브라우저가 무엇을 들고 있는지, API를 누가 호출하는지, 서버가 무엇을 저장하는지, 무엇을 검증하는지를 네 구조에 똑같이 물었습니다.

	SPA 브라우저가 직접 호출	Token Mediator mediator가 토큰 보관	BFF 서버가 대신 호출	Forward-Auth edge가 신원 확정
브라우저 보관	access + refresh	access token만	없음	없음
API 호출자	브라우저	브라우저	BFF 서버	Edge proxy
서버 상태	없음	authorized client	session + client	proxy session
CSRF 경계	Bearer header	Bearer header	XSRF token	proxy session
검증 대상	Bearer JWT	Bearer JWT	Bearer JWT	trusted header

브라우저에서 token을 치우면 서버 쪽에 세션 상태가 생기고, 세션 쿠키를 쓰면 CSRF 경계가 따라옵니다. 이 표는 네 구조를 모두 구현해 본 실험 결과이고, 이 가운데 두 가지를 TechLog에 실제로 넣었습니다.

10 · TechLog · Authentication Modes

TechLog에는 `jwt`와 `redis-session` 두 mode를 구현했습니다

같은 endpoint인데 상태는 어디에 남는가? · TechLog Backend · `SecurityConfig` runtime branch

`SecurityConfig`는 `auth-mode` 값에 따라 두 SecurityFilterChain 중 하나를 고릅니다. 두 branch는 같은 endpoint를 서비스하지만 상태가 남는 위치가 다릅니다.

	auth-mode = jwt	auth-mode = redis-session
브라우저가 가진 것	Bearer JWT	session cookie + XSRF token
서버가 저장하는 것	없음 · STATELESS	HttpSession → Redis
무엇을 검증하는가	JWT 서명	server session + CSRF token
logout	토큰 만료까지 유효	session invalidate + cookie 삭제

`jwt`는 Resource Server가 서명만 검증하고 서버에 session을 만들지 않습니다. `redis-session`은 `oauth2Login`으로 session을 만들고 SecurityContext를 Redis에 보관하며, session cookie를 쓰는 만큼 CSRF token 경계가 함께 붙습니다. repository 기본값과 현재 로컬 `.env`는 `jwt`이고, 공개 배포가 어느 mode로 떠 있는지는 이 작업공간에서 확인되지 않아 단정하지 않습니다. Forward-Auth는 비교 실험 범위이고 TechLog security branch에는 넣지 않았습니다.