이승현

전화 | 010-9797-6717 이메일 | fhzktm@naver.com 블로그 | shnowball.tistory.com

자기소개

기술을 배우는 것도 즐겁지만, 눈앞의 불편을 없애는 일이 더 가치있게 느껴집니다.

새로 배운 것을 어디에 쓸지가 정해지지 않으면 손이 오래 가지 않았습니다. 반대로 여행 사진이 카카오톡과 구글 드라이브로 흩어져 아무도 다시 찾아보지 않게 되는 일, 매주 회고로 찾아낸 개선점이 그날로 끝나버리는 일처럼 제가 직접 겪어 불편했던 것들은 계속 붙잡고 있게 됐습니다. 아래 두 개인 프로젝트도 그렇게 시작했습니다.

만들고 나서가 더 어려웠습니다.

가족 8명에게 배포하고 나서 "동작은 하는데 순서가 불편하다"는 말을 들었습니다. 제 눈에는 괜찮아 보이던 화면이었는데, 실제로 쓰는 사람에게는 단계가 하나 더 있다는 것만으로 안 쓰게 되는 이유가 됐습니다. 기능을 더하는 대신 화면 구조를 두 차례 다시 짰고, 만든 사람은 자기 제품의 불편을 잘 보지 못한다는 것을 그때 배웠습니다.

AI로 확보한 시간은 판단에 투자하려 합니다.

AI와 함께 작업한 덕분에 2~3일 만에 프로토타입을 배포할 수 있었고, 그만큼 무엇을 만들고 무엇을 버릴지 고민할 시간이 늘었습니다. 다만 성능이 느리거나 버그가 났을 때 원인까지 맡기지는 않으려 합니다. 조회가 느렸을 때도 짐작으로 고치는 대신 구간을 나눠 다시 재본 다음에야 원인이 두 겹이었다는 것을 알 수 있었습니다.

프로젝트

AI Native 개발 환경

2026.07 ~ 진행 중

개인 작업 환경

여러 사이드 프로젝트를 같은 기준으로 기획·설계·기록할 수 있도록, 개인 지식 베이스와 AI 에이전트 운영 규칙을 직접 설계한 작업 환경

역할: 계층 구조 설계, 운영 규칙 정의, 반복 작업 절차 문서화

문제

- 사이드 프로젝트를 진행하며 AI와 함께 작업했는데, 같은 것을 물어도 답이 매번 달라지고 앞서 내린 결정이 다음 작업으로 이어지지 않음
- 결정의 이유가 대화 안에만 남아, 시간이 지나면 왜 그렇게 정했는지 확인할 방법이 없음

→ AI를 더 잘 쓰는 것보다, AI가 매번 같은 기준으로 일하도록 환경을 만드는 것이 먼저라고 판단

- **지식이 쌓이도록 3계층 구조로 분할**
  Andrej Karpathy가 공개한 LLM Wiki 패턴을 참조해 원본 자료 · 원본을 종합한 문서 · 그 구조를 정의한 스키마로 계층을 나누고, 제 상황에 맞게 도메인 구분과 페이지 규칙을 구체화. 새 자료가 들어오면 매번 처음부터 다시 찾는 대신 종합 문서 자체를 갱신하게 해, 자료가 늘어도 찾는 비용이 함께 늘지 않도록 함. 원본은 수정하지 않는 규칙을 두어 출처 추적이 끊기지 않게 함
- **반복되는 작업을 절차 문서로 표준화**
  프로젝트 착수, 설계 결정 기록, 진행 로그 갱신처럼 반복되는 작업을 각각 절차 문서로 정의해, 언제 실행해도 같은 형식과 기준으로 결과가 나오도록 함. 스택이 매번 달라지는 구현 작업은 범용 절차로 묶지 않고 프로젝트 전용 절차로 분리
- **지적받은 것을 규칙으로 승격시키는 루프**
  잘못된 판단을 지적받으면 그때만 고치고 끝내지 않고 **무엇을 · 왜 틀렸는지 · 다음엔 어떻게 할지**를 규칙 문서에 남기고, 작업 시작 시 반드시 읽도록 절차에 넣음. 같은 실수가 그래도 반복되면 규칙의 적용 조건을 더 좁혀 다시 씀(예: "오래된 것 같으면 날짜를 확인한다" → "날짜를 쓰기 직전에 무조건 확인한다")
- **결과**
  가족 아카이브를 진행하는 동안 설계 결정과 그 배경이 이 구조로 쌓이면서, 이후 작업에서 "그때 왜 그렇게 정했는지"를 다시 묻지 않고 이어갈 수 있게 됨. 새 프로젝트를 시작할 때도 같은 절차와 형식을 그대로 재사용하고 있음

커리어 나침반

2026.01 ~ 진행 중

개인 프로젝트

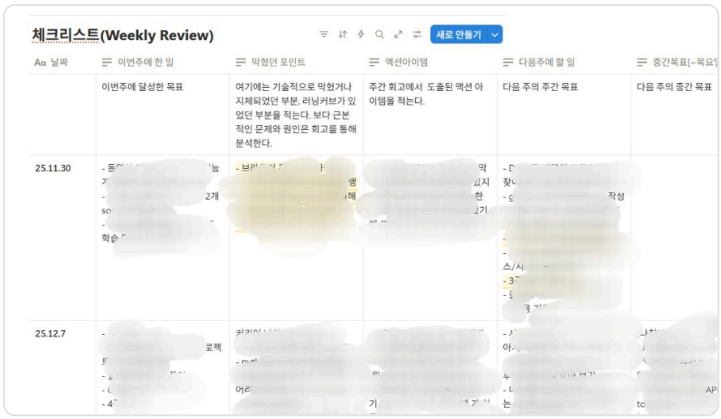
매일의 실행 기록과 주간 회고를 계속 쌓아, 쌓일수록 더 정확해지는 개인화 피드백을 주는 AI 어시스턴트

역할: 서비스 기획, RAG 파이프라인 구축, 로컬 LLM 커스터마이징, 옵저버빌리티 환경 구축 (Java, Spring Boot, Spring AI, Ollama/gemma3, Zipkin)

문제

- 주간 회고 노션 문서로 매주 문제와 개선점을 찾아냈지만, 그 결과가 회고 시점에서 끝나고 이후 피드백에 자료로 이어지지 않음
- 대화형 AI는 개인 컨텍스트를 반영하지 못해 일반적인 피드백에 그침

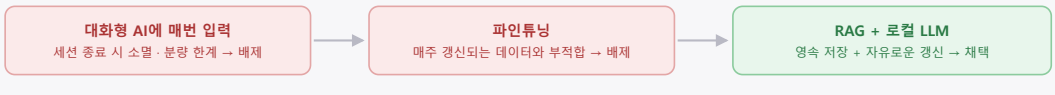
→ 오늘 한 일, 오늘의 목표, 지난 회고를 한자리에 놓고 판단하는 기록이 쌓일수록 정확해지는 피드백 파이프라인이 필요했음



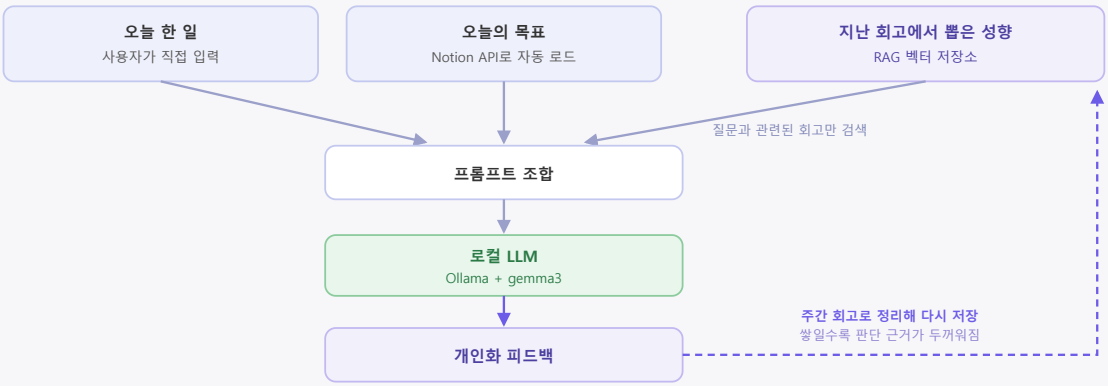
주간 회고 체크리스트 (내용은 비공개 처리)

① 제품 기획

- **오늘의 기록이 다음 판단의 재료가 되도록 순환 구조를 설계**
  오늘 한 일을 입력하면 노션에 적어둔 그날의 목표와 대조하고, 지난 회고에서 뽑아낸 성향 정보를 함께 참고해 피드백을 만듦. 그 피드백을 반영해 쓴 회고가 다시 저장소에 쌓여 다음 피드백의 재료가 되므로, 쓸수록 판단 근거가 두꺼워지는 구조
- **방법론 세 가지를 검토해 선택**
  대화형 AI에 매번 회고를 붙여넣는 방식은 세션이 끝나면 컨텍스트가 사라지고 한 번에 넣을 분량에도 한계가 있어 배제. 파인튜닝은 회고가 매주 추가·수정되는 성격이라 한 번 학습시키는 방식과 맞지 않아 배제. 영속 저장과 자유로운 갱신을 모두 만족하는 RAG와 로컬 LLM 조합을 택함
- **기록의 원천을 일원화**
  목표를 서비스에 다시 입력받는 대신 Notion API로 가져옴. 이미 노션으로 일원화된 관리 원천을 다원화하지 않고, 옮겨 적는 반복 작업만 줄임
- **실사용으로 효용 확인**
  회고에서 반복되던 벼락치기 습관을 서비스가 먼저 짚어주는 경험을 하며 계속 쓸 만하다고 판단



방법론 선택 과정: 무엇을 왜 배제하고 무엇을 택했는가



처리 흐름: 세 가지 입력이 하나의 피드백이 되고, 그 결과가 회고로 정리돼 다시 저장소로 돌아온다

② 엔지니어링 · 환각 해결

모델이 지난 회고에 적힌 사건을 오늘 일어난 일로 착각하는 환각이 반복됐습니다. 예를 들어 지난주 회고에 "도커 컴포즈를 적용하지 않아 이슈 추적이 어려웠다"고 적어둔 상태에서 오늘 "도커 파일을 수정했다"고 입력하면, 지난주 문제를 오늘의 문제인 것처럼 답했습니다.

- **1차 시도 실패 후 가정을 재확인**
  프롬프트에 "이 내용은 과거 기록이니 오늘 입력과 혼동하지 말 것"이라는 지시문을 추가했지만 효과가 크지 않았음. 지시문을 여러 번 고쳐도 나아지지 않자 "프롬프트만 고치면 된다"는 처음의 가정을 의심하게 됨
- **고치기 전에 관측 환경을 먼저 구성**
  - RAG의 검색과 프롬프트 조합이 프레임워크(Spring AI) 내부에 감춰져 있어, 모델에 실제로 전달되는 프롬프트 전문을 확인할 방법이 없었음
  - 일반 로그로는 요청 한 건이 검색 → 조합 → 모델 호출로 이어지는 흐름을 하나로 묶어 볼 수 없었음
  - Zipkin으로 각 구간을 추적해 **모델에 전달되는 프롬프트 전문과 검색된 회고를 눈으로 확인**할 수 있는 환경을 먼저 만듦
- **트레이싱 기록을 근거로 네 가지를 순차적으로 개선**
  - 1 검색된 회고가 날짜 구분 없이 원문 그대로 삽입되고 있었음 → 저장 형태를 사건 서술("~했다")이 아닌 성향 서술("이 사람은 ~한 편이다")로 가공. 컨텍스트와 오늘 입력이 문장 형태에서부터 구분됨
  - 2 시점을 혼동하지 말라는 지시만 있고 판단 기준이 없었음 → 청크마다 회고 작성일을 붙여 비교 대상을 만들고, 시스템 메시지로 오늘 날짜와 대조하도록 지시
  - 3 질문과 관련이 옅은 회고까지 함께 딸려오고 있었음 → 한 번에 가져오는 검색 개수(top-k)를 줄임
  - 4 한 덩어리 안에 여러 시점의 내용이 섞여 있었음 → 회고를 자르는 크기와 분할 기준을 조정해 한 덩어리가 한 맥락만 담도록 변경
- **결과**

환각 발생이 눈에 띄게 줄었음(정량 측정은 하지 않음). 고질적으로 반복되던 문제 패턴을 서비스가 먼저 짚어주는 피드백을 받게 됐고, 이를 계기로 회고 방향성 자체를 다시 잡음

환각 해결 과정: 고치기 전에 볼 수 있게 만들고, 관측 기록을 근거로 네 가지를 순차 개선

가족 아카이브

2026.07 ~ 2026.08

개인 프로젝트

여러 플랫폼에 흩어진 가족 사진을 한 곳에 모아 여행·행사 단위 앨범으로 정리하고, 가족 8명이 실제로 쓰도록 반복 개선한 사진 아카이브 서비스

역할: 기획·개발·배포·운영 전체 담당 (Next.js, Cloudflare Workers/R2, Supabase, Google Analytics)

개발기 **기획부터 배포·성능 개선까지 전 과정 기록 · shnowball.tistory.com/223**

문제

- 주기적으로 여행을 다니는 가족이라 사진이 계속 쌓이는데, 저장되는 곳은 카카오톡 단톡방·구글 드라이브·개인 기기로 매번 달라짐. 어느 플랫폼에 무엇이 있는지 아무도 알지 못하고, 한 곳에 모으는 주체도 없음

→ **사진이 여러 플랫폼으로 분산되는 것을 막고 한 곳에 모으는 것을 목표로, 여행·행사 단위로 묶이는 가족 전용 저장소를 기획**

① 제품 기획

- **카카오톡 자동 연동을 조사한 뒤 기각하고 수동 방식으로 확정**
  사진 대부분이 단톡방에 있어 자동 수집을 먼저 검토했으나 정식 API가 없었음. 비공식 경로로 자동화를 강행하는 것보다, 관리자가 대화내보내기를 주기적으로 실행하는 단순한 절차가 낫다고 판단
- **되돌릴 수 없는 동작 하나만 관리자로 제한**
  초기 설계는 "업로드는 관리자만"이었으나, 실제로 복구가 불가능한 동작은 삭제뿐이라고 보고 삭제만 관리자 전용으로 좁히고 업로드·앨범 정리는 가족 전체에 개방
- **자동 분류와 사람의 확인을 함께 쓰는 하이브리드 분류**
  사진의 촬영일자(EXIF)를 읽어 3일 이내 연속 촬영을 한 앨범으로 자동 묶고, 관리자가 2차로 확인·보정. 카카오톡 압축 전송으로 촬영 정보가 사라진 사진은 잘못 묶이는 편보다 낫다고 보고 자동 분류 대상에서 제외해 수동으로 넘김
- **MVP 범위를 좁게 제한**
  태그·인물 검색과 개인별 로그인은 배제하고, 비공개 링크와 공유 비밀번호만으로 접근을 단순화

② 엔지니어링 · 문제 해결

- **앨범 조회 속도: 첫 응답 1.4~3.9초 → 약 0.19초, 사진 1장 4.76MB → 46KB**
  - 증상 앨범을 열 때마다 화면이 뜨기까지 몇 초씩 걸림
  - 측정 실제 서비스에 요청을 보내 구간을 나눠 재본 결과, 원인이 두 겹으로 나뉨
  - 원인 1 서버가 조회할 때마다 저장소(R2)에서 사진 파일을 다시 열어 가로·세로 크기를 계산하고 있었음
  - 원인 2 목록의 작은 썸네일 자리에 원본 사진(장당 4.76MB)을 그대로 내려받고 있었음
  - 해결 크기는 업로드 시점에 한 번만 계산해 DB에 저장하고, 썸네일은 업로드 직전 브라우저에서 만들어 함께 저장
  - 검증 측정값에 매 요청마다 새로 맺는 TCP·TLS 연결 비용 약 0.6초가 섞여 있는 것을 발견하고, 실제 브라우저처럼 연결을 재사용하는 조건으로 다시 재서 수치를 확정
- **모바일에서 화면 오른쪽이 잘리는 버그: 증상만 막았다가 재발**
  - 1차 조치는 넘치는 영역을 잘라내는 CSS로 증상만 가렸고, 다른 화면에서 그대로 재발
  - 다시 찾아본 원인은 긴 제목을 한 줄로 고정하는 CSS 속성이었음. 이 속성 때문에 모바일 브라우저가 최초 렌더링 시 화면 폭 자체를 잘못 계산하고 있었음
  - 줄바꿈이 가능한 방식으로 교체하고, 코드 전체에 같은 속성이 남아있지 않은지 확인
- **사진 확대 화면의 관리 메뉴 버그: 하나의 원인으로 5가지 증상 해결**
  - 메뉴가 사진에 가려짐, 클릭이 사진 쪽으로 먹힘, 확대를 닫아야만 보임 등 증상이 제각각이었음
  - 사용 중인 확대 라이브러리의 CSS를 직접 확인해 화면 겹침 순서 값이 10배 차이 나는 것을 발견, 값을 조정해 5가지 증상을 한 번에 해소
- **DB 연결 실패를 제약으로 두지 않고 원인을 규명**
  스키마 변경 스크립트가 반복 실패해 우회를 검토했으나, 다시 조사해 보니 DB 호스트가 IPv6 주소만 가진 것이 원인이었음. IPv4로 접속 가능한 커넥션 풀러로 연결해 해결

③ 유저 피드백 루프 · 사용 데이터 관측

- **"동작은 하는데 순서가 불편하다" → 화면 구조를 단일 경로로 다시 설계**
  정리 화면과 업로드 동선이 나뉘어 있고 사진 확대까지 단계가 많다는 의견, 첫 화면에서 선택지부터 마주치는 것이 부담이라는 의견을 받음. 기능을 더하는 대신 **탭과 선택지를 없애고 앨범 안에서 바로 업로드·즉시 확대되는 하나의 경로**로 다시 설계

- **"앨범마다 추억을 알려줬으면 좋겠다" → 기념일 알림(웹 푸시) 구현**
  앨범의 첫 촬영일과 월·일이 같은 날 오전 9시에 기기 알림을 매년 발송. 스케줄러를 본체와 분리된 별도 Worker로 두는 과정에서 같은 계정의 Worker끼리 호출이 무한 루프 방지 정책에 막히는 문제를 겪었고, 설정으로 해결해 실제 기기에서 알림 수신까지 확인
- **말로 듣는 피드백만으로는 안 보이는 것이 있어 사용 데이터를 추적**
  - 한계 가족들이 불편한 점은 말해줬지만, 얼마나 자주 들어오는지·어느 단계에서 멈추는지는 물어봐도 정확히 알기 어려웠음
  - 계측 Google Analytics를 붙이고 방문·세션 같은 기본 지표에 더해 **앨범 진입 · 사진 확대 · 업로드 시작 · 업로드 완료**를 각각 커스텀 이벤트로 심어, 접속부터 업로드까지의 퍼널과 재방문 간격을 볼 수 있게 함
  - 해석 사용자가 8명뿐이라 전환율 같은 비율 지표는 한 사람의 행동에 크게 흔들림. 비율 대신 **행동별 절대 횟수와 개별 세션의 이동 경로, 재방문 간격**을 기준으로 읽음
  - 발견 전체 가족 중 일부만 반복해서 들어오고 있었고, **여행을 다녀온 뒤에도 업로드가 일어나지 않는 구간이 반복**됨
  - 원인 왜 안 쓰는지는 수치만으로 알 수 없어 가족에게 직접 물어봄. 앱에 들어갈 계기 자체가 없다는 답을 얻음
  - 개선 여행 이후 업로드를 유도하는 알림과 앨범별 주기 알림을 추가해 배포. 기념일 알림 하나에 그쳤던 알림 기능을 사용 데이터를 근거로 확장
- 결과: 말로 받은 피드백과 사용 데이터를 함께 보며 재설계와 재배포를 반복하고, 가족 8명이 계속 쓰는 상태로 운영

MVPQuest

2024.07 ~ 2024.08

팀 프로젝트

사내 QA만으로는 커버하기 어려운 다양한 사용자 환경의 결함을 보완하기 위해, 불특정 다수 사용자가 신규 MVP·기능을 직접 테스트하는 크라우드소싱 기반 검증 플랫폼

역할: 코어 도메인 "mvptest" 개발 및 테스트 작성, API 성능 최적화, 인프라 구축

- **도메인 로직을 프레임워크에서 분리**
  핵심 비즈니스 규칙을 Spring·JPA에 의존하지 않는 순수 객체로 두어, 프레임워크가 없어도 규칙 자체가 성립하도록 구성. 도메인 변경이 인프라 코드로 번지지 않고, 객체를 생성하는 것만으로 규칙을 검증할 수 있게 됨
- **검증을 통합 테스트에서 단위 테스트로 옮겨 실행 시간 단축**
  도메인 규칙까지 스프링 컨텍스트를 띄우는 통합 테스트로 확인하고 있어, 테스트 한 건마다 컨텍스트 로딩 비용을 지불하고 있었음. 위에서 분리한 순수 도메인 객체 덕분에 컨텍스트 없이 검증할 수 있게 되어 단위 테스트로 재구성했고, 경계값과 예외 경로까지 촘촘히 덮으면서도 전체 테스트 수행 시간을 줄임
- **커서 기반 페이지네이션 도입**
  Offset 방식은 뒤쪽 페이지로 갈수록 앞의 행을 모두 훑어야 해 데이터가 커질수록 느려지고, 조회 도중 새 데이터가 삽입되면 항목이 밀려 중복·누락이 발생. 정렬 기준과 PK를 묶은 복합 커서(sortDate, id)로 전환해 같은 날짜의 순서 충돌을 PK로 해소하고, 데이터 양과 무관하게 일정한 조회 성능을 확보. 목록 조회의 N+1은 Batch Fetching으로 완화
- **단건 조회 캐싱: 평균 응답시간 350ms → 80ms (약 77% 개선)**
  - 문제 존재하지 않는 데이터를 조회하면 캐시에 저장할 값이 없어 매번 캐시를 통과해 DB까지 도달함. 없는 키로 요청이 반복되면 캐시가 있어도 DB 부하가 그대로 몰리는 구조였음(Cache Penetration)
  - 해결 조회 결과가 없는 경우에도 빈 값을 캐싱해 동일 키의 반복 요청이 DB에 닿지 않도록 차단
  - 문제 삭제된 데이터가 캐시에 남아 정상 응답으로 내려감. 값이 오래된 것과 이미 없어진 것은 다른 문제라고 판단
  - 해결 삭제는 캐시를 즉시 무효화하고, 갱신은 TTL 자연 만료에 맡기도록 정책을 분리
- **소셜 로그인 확장 구조**
  OAuth2 클라이언트로 네이버·구글 로그인을 구현하고 Strategy 패턴을 적용해, 클라이언트 코드 수정 없이 인증 공급자를 추가할 수 있도록 구성
- **무중단 배포: 배포 다운타임 3분 → 0분**
  Docker 기반 Blue/Green 배포를 구축하고 GitHub Actions CI/CD 파이프라인에 통합해, 코드 통합마다 병목이던 빌드·테스트를 자동화
- **모니터링과 장애 감지**
  Prometheus + Grafana로 Redis 캐시 히트율과 서버 CPU·메모리를 관측하고, 임계치(80%) 초과 시 팀 디스코드로 웹훅 알림을 전송. Redis Sentinel failover를 구성해 단일 노드 장애가 전체 장애로 번지지 않도록 대비

Coupong

2024.07

팀 프로젝트

선착순 쿠폰 발급에서 발생하는 중복 발급과 재고 오차를 해결하며, 동시성 제어를 락의 적용 범위 순으로 단계적으로 학습한 프로젝트

역할: 동시 요청 테스트 시나리오 구현, 경합 상태 해결, 쿠폰 발급 API 성능 개선

정리 글 **DB 락과 Redis 락(Lettuce · Redisson) 비교 · shnowball.tistory.com/118**

문제

- 수천 건의 요청이 동시에 몰리는 선착순 이벤트에서 같은 쿠폰이 중복 발급되고 재고 수량에 오차가 발생

→ "락을 걸면 된다"에서 끝내지 않고, 락이 유효한 범위가 어디까지인지를 기준으로 세 단계를 직접 구현하며 확인

① 락의 적용 범위를 넓혀가며 단계적으로 구현

- **1단계 · JVM 내 락 (synchronized, ReentrantLock)**
  두 방식을 모두 적용해 비교. 한 서버 안에서는 정합성이 지켜지지만 락의 유효 범위가 JVM 하나로 한정되어, 인스턴스를 늘리면 서버마다 별도의 락이 생겨 서로를 막지 못함
- **2단계 · DB 비관적 락**
  여러 인스턴스가 같은 DB를 바라보므로 인스턴스 간 정합성은 확보. 다만 잠금 대상이 DB의 행에 한정되어, 재고나 발급 이력처럼 **잠가야 할 상태가 DB 밖에도 걸쳐 있으면 임계 영역 전체를 감싸지 못함**. 고부하에서는 데드락과 커넥션 고갈 위험도 함께 확인
- **3단계 · Redis 분산 락**
  저장소가 무엇이든 하나의 키로 임계 영역 전체를 잠글 수 있어, 여러 인스턴스와 여러 저장소가 얽힌 상황에 맞는 방식으로 채택

② 전환 후에도 중복 발급이 남았던 이유

- **락과 트랜잭션의 경계가 어긋나 있었음**
  Redis 락으로 바꾼 뒤에도 중복 발급이 계속 발생. 증상이 아니라 구조를 다시 보니 트랜잭션이 커밋되기 전에 락이 먼저 반납되고 있었고, 그 사이에 다른 스레드가 진입해 아직 반영되지 않은 재고를 읽고 있었음

- **커스텀 Transactional Advice로 순서를 보장**
  트랜잭션 범위를 명시적으로 지정하고, 트랜잭션이 완전히 종료된 뒤에 락을 반납하도록 순서를 고정해 중복 발급을 없앰
- **락 획득 방식 개선: 1,000건 동시 요청 기준 평균 응답시간 4,400ms → 1,800ms (약 59% 개선)**
  대기 중인 스레드가 계속 획득을 재시도하는 스핀락 방식이 Redis에 폴링 부하를 준다는 점을 확인하고, 락 해제 메시지를 받은 시점에만 재시도하는 Redisson의 pub-sub 기반 락으로 교체
- **검증**
  멀티 스레드 동시 요청 테스트 시나리오를 직접 구현해 각 단계마다 쿠폰 재고와 실제 발급 수량의 오차를 측정

학력

인천대학교 2019.02 ~ 2023.02

학사 | 임베디드 시스템 공학과 | 졸업

자격증

SQLD (SQL 개발자)

기술 스택

BackEnd	Java, Kotlin, Spring Boot, Spring Data JPA, QueryDSL, Spring AI, MySQL, PostgreSQL, Redis
FrontEnd	JavaScript, TypeScript, Next.js
DevOps	Docker, GitHub Actions, Nginx, EC2, Cloudflare Workers/R2, Supabase
Observability	Prometheus, Grafana, Zipkin, Google Analytics
AI	Ollama(gemma3), RAG (벡터 검색 · 임베딩)