


포트폴리오1: 🛒 이커머스 서비스 서버

개인 프로젝트

2024.10-2024.12

 <https://github.com/jimyungkoh/ecommerce-service-server>

문제와 해결 ⭐

문제: 대용량 상품 조회 성능 문제 해결 (1,000만 건)

해결 과정:

신상품순 조회(상품 수정일) 데이터 특성 분석

데이터 시딩 기준: 2010-01-01 ~ 현재까지의 기간; 밀리초[datetime(3)] 단위

특징: 카디널리티 1로 각 상품의 수정 시간이 거의 유일한 값을 가짐

성능 영향: 높은 카디널리티로 인해 인덱스 효율성 극대화 예상

```
CREATE INDEX idx_product_updated_at ON product (updated_at);
```

가격순 조회(가격) 데이터 특성 분석

데이터 시딩 기준: 1,000원 ~ 10,000,000원까지의 값; 100원 단위

특징: 카디널리티 0.01로 같은 가격대의 상품이 한 가격당 약 100개씩 존재함

성능 영향: 카디널리티가 낮더라도 값이 정렬 상태로 저장되므로 쿼리 성능이 크게 향상될 것으로 예상됨

```
CREATE INDEX idx_product_price ON product (price);
```

결과:

신상품순 조회

개선 전	5.01초	개선 후	0.000154초
<div>Table scan → Sort → Limit</div> <div>스캔 행 수: 10,000,000</div> <div>정렬 방식: 전체 데이터 정렬</div>		<div>Index scan → Limit</div> <div>스캔 행 수: 20</div> <div>정렬 방식: 인덱스 정렬 활용</div>	

가격순 조회

개선 전	8.22초	개선 후 (가격 낮은순) ↑	0.015초	개선 후 (가격 높은순) ↓	0.014초
<div>Table scan → Sort → Limit</div> <div>스캔 행 수: 10,000,000</div> <div>예상 비용: 1.05e+6</div> <div>실제 시간: 8220ms</div>		<div>Index scan → Limit</div> <div>스캔 행 수: 20</div> <div>예상 비용: 0.121</div> <div>실제 시간: 14.9ms</div>		<div>Index scan (reverse) → Limit</div> <div>스캔 행 수: 20</div> <div>예상 비용: 0.121</div> <div>실제 시간: 13.7ms</div>	

문제: 프로모션 시나리오에서 동시 주문(초당 1,000건)으로 재고 초과 주문과 낮은 처리량 문제 발생

해결 과정:

재고 초과 주문 문제

- 원인
 - 동시다발적 주문 요청으로 인한 데이터 정합성 깨짐
 - 트랜잭션 격리 수준 부적절
- 해결 방안: 데이터 정합성 확보
 - 재고 테이블에 row-level lock 적용
 - 비관적 락(Pessimistic Lock) 도입
 - 순차적 처리로 초과 주문 원천 차단

낮은 처리량 문제

- 원인
 - 단일 트랜잭션 내 과도한 처리 로직
 - 동기식 API 호출로 인한 응답 지연
- 해결 방안: 이벤트 기반 트랜잭션 분리
 - 주문 프로세스 단계 분리: 주문 생성 → 재고 감소(이벤트) → 결제(이벤트) → 주문 완료(이벤트)
- 비동기 처리 도입
- 서비스 간 결합도 감소

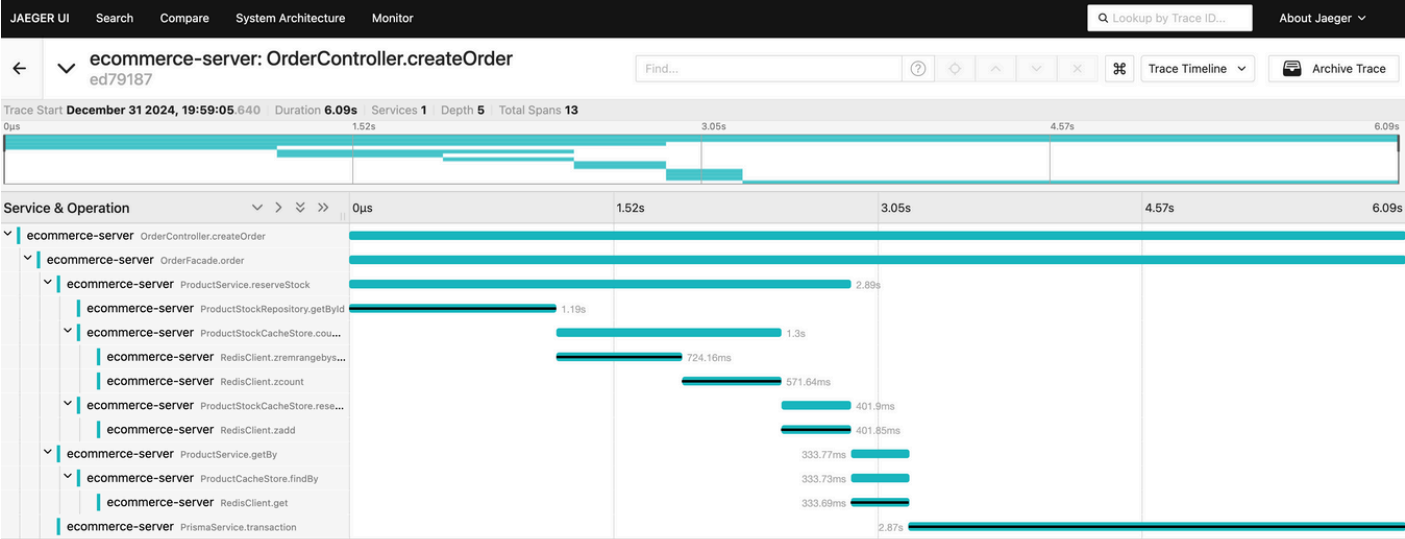
결과:

- 평균 응답 시간 51.6% 단축 (5.19초 → 2.51초)
- 요청 처리량 96% 향상 (177/s → 347/s)

기타 성과

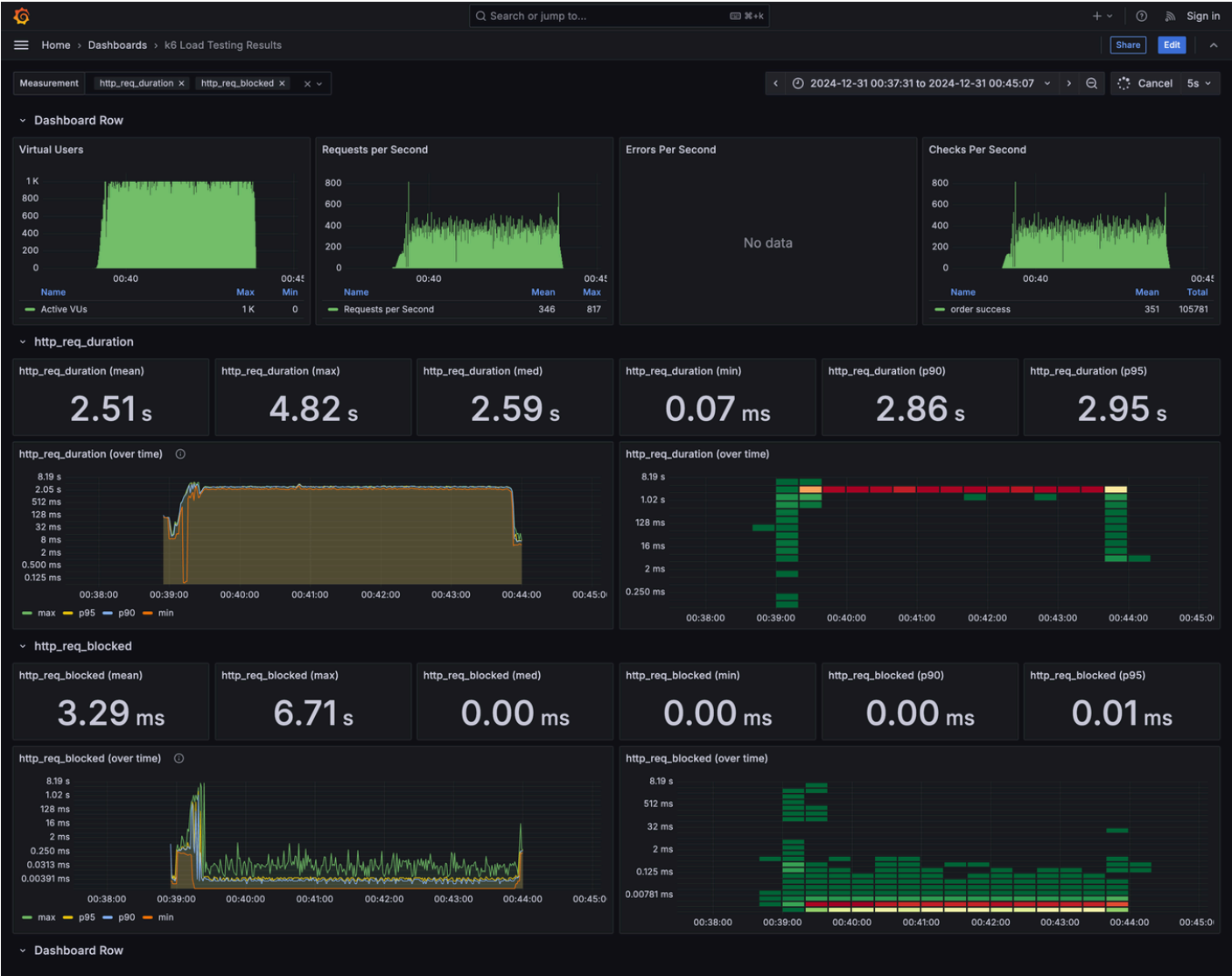
분산 추적 시스템 구축

비동기 장애 추적이 어려웠던 주문-재고-결제 플로우에 OpenTelemetry 기반 추적 시스템을 구축하여 응답시간 프로파일링 Trace ID 기반으로 5계층 13개 트랜잭션의 의존성을 시각화하여 장애 원인 분석 시간 단축



부하 테스트 모니터링 환경 구축 (Grafana - InfluxDB - k6)

실제 프로덕션 환경의 트래픽 패턴을 시뮬레이션하기 위해 k6로 부하 테스트 자동화
시계열 데이터베이스(InfluxDB)로 테스트 결과를 저장하여 성능 변화 추이와 패턴 분석 가능
주요 성능 지표(VUser, RPS, 응답시간)를 Grafana로 실시간 모니터링하여 즉각적인 병목 구간 감지



팀프로젝트

사진 방명록 공유 플랫폼
2024.05 -2024.09


역할

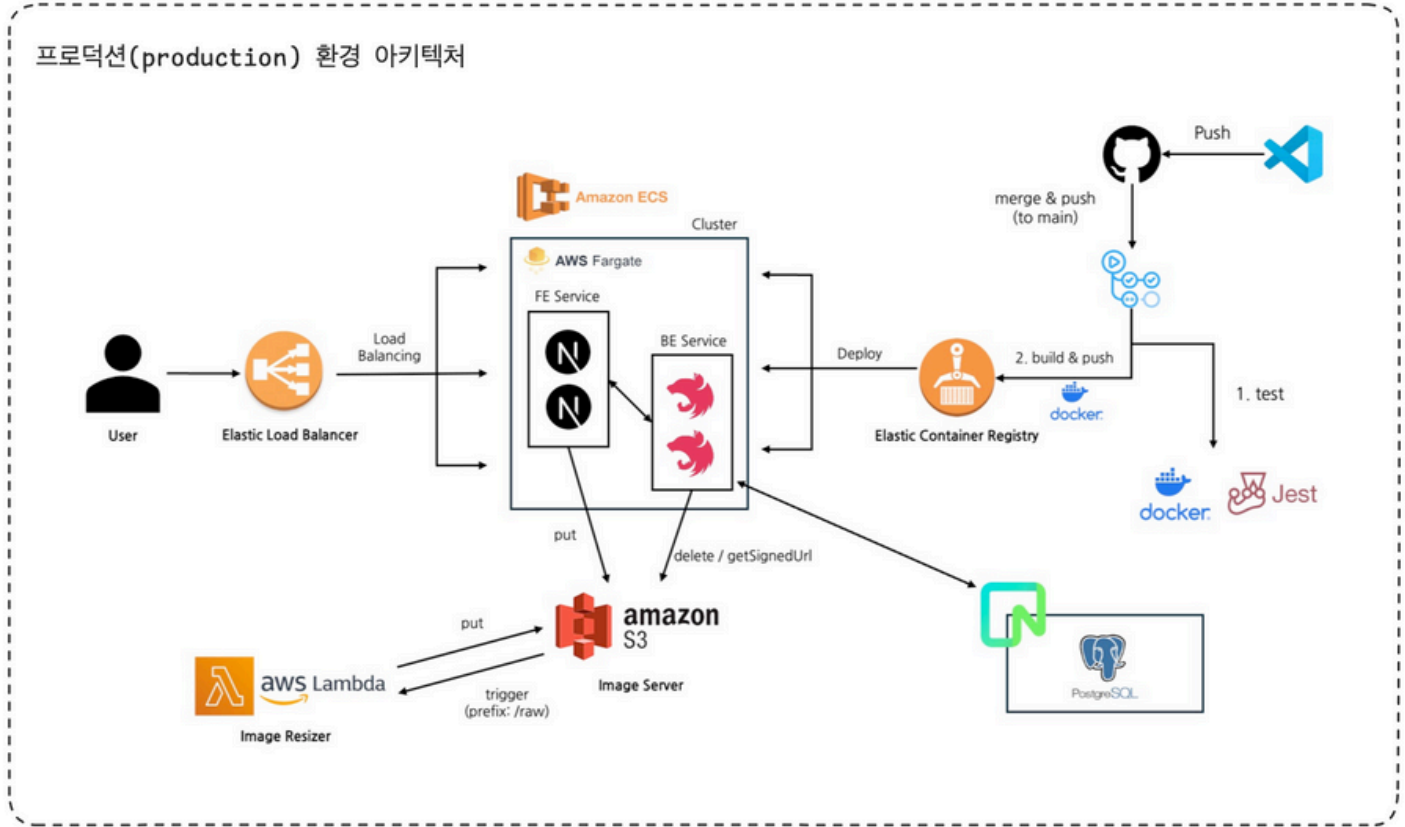
Client	<ul style="list-style-type: none">방명록(링크 생성, 작성, 상세) 페이지공통 컴포넌트 설계폴라로이드 사진 필터 구현
Server	<ul style="list-style-type: none">방명록 도메인 API 설계 및 구현이미지 최적화 시스템 구축AWS 클라우드 인프라 구축 및 서비스 배포·운영



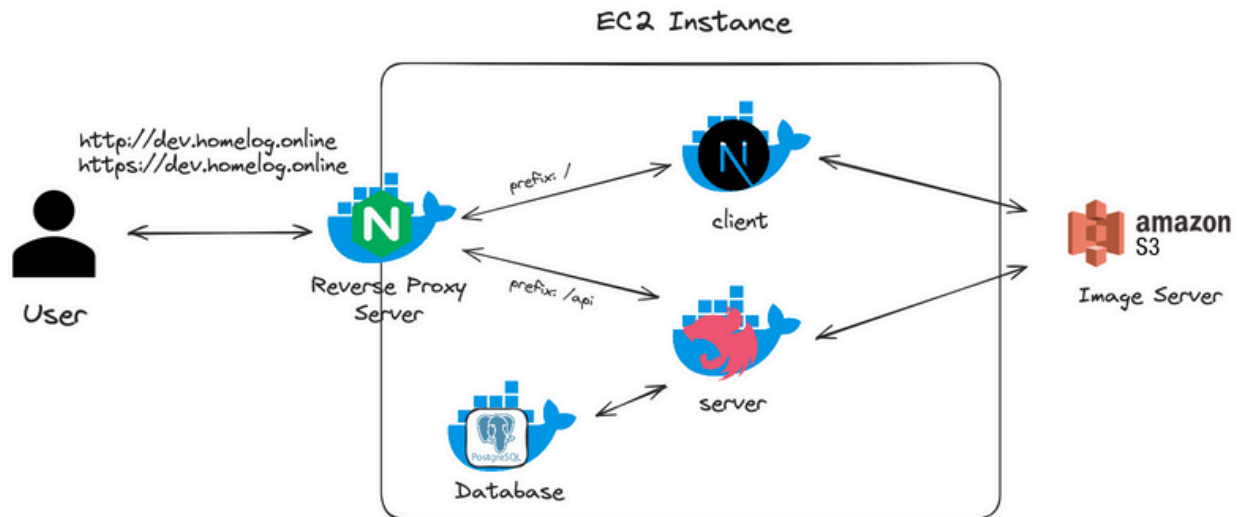
타임스탬프 필터를 지원하는 사진 기반 방명록 웹

클라이언트  <https://github.com/jimyungkoh/homelog-client>

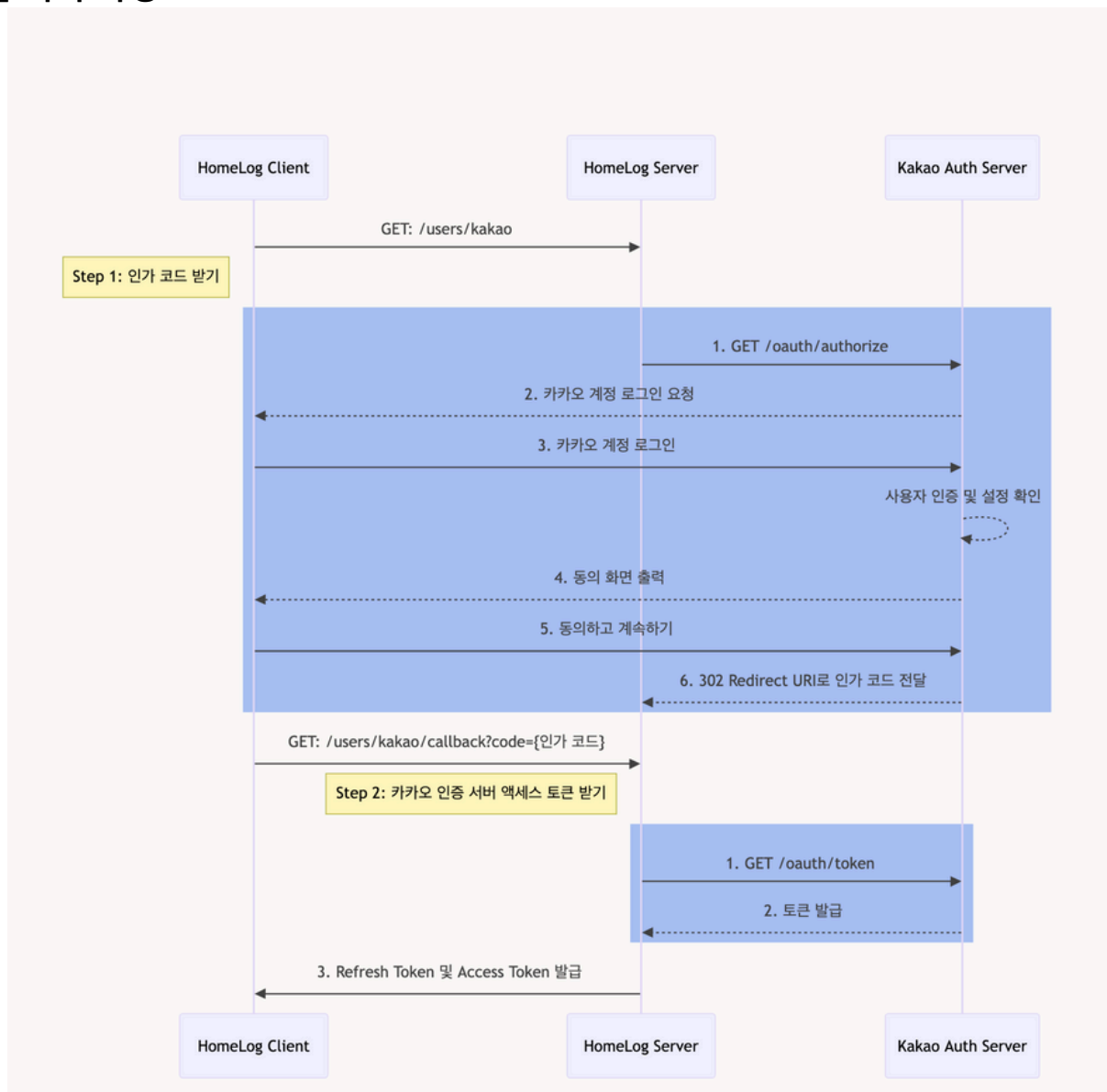
서버  <https://github.com/jimyungkoh/homelog-server>



개발(dev) 환경 아키텍처



로그인 처리 과정



문제와 해결 🌟

문제: 개발 환경 배포 비용 최적화

- 상황: 프로덕션과 동일한 ECS, ALB 구성으로 인해 불필요한 비용이 발생하고 있음
- 영향: 월 고정비용이 \$65로 증가하여 개발 비용이 상승
- 세부사항: 24/7 운영이 필요 없는 개발 환경에서 고가용성 인프라를 사용 중

해결과정:

- 비용 분석
 - ECS 클러스터 운영 비용 분석
 - ALB 사용량 대비 비용 분석
 - 개발 환경의 특성 파악 (간헐적 사용, 고가용성 불필요)
- 대안 검토
 - EC2 온디맨드와 Spot 인스턴스 비용 비교
 - 로드밸런서 대안으로 Nginx reverse proxy 검토
 - 개발 환경에 적합한 구성 설계
- 구현
 - EC2 Spot 인스턴스 구성
 - Nginx reverse proxy 설정
 - 자동화된 환경 구성 스크립트 작성

결과:

- 프로덕션과 유사한 개발 환경 인프라 구성 달성
- 월 고정비용 77% 절감 (\$65 → \$15)

문제: 브라우저 호환성으로 인한 이미지 처리 오류

- 상황: Safari 브라우저 및 iOS 인앱 브라우저에서 이미지 필터 적용 후 저장 기능 동작 실패

해결과정:

- 원인 분석
 - 이미지 캡처 라이브러리 호환성 조사(깃허브 이슈 분석)
 - Safari에서의 이미지 처리 특이사항 확인: DOM 캡처시 첫번째 시도에서 이미지가 누락됨
- 대안 검토
 - 다른 이미지 캡처 라이브러리 검토
 - 브라우저 네이티브 API 조사
- 구현
 - Canvas API를 사용한 이미지 처리 로직 개발
 - 필터 효과를 Canvas 렌더링으로 구현

결과:

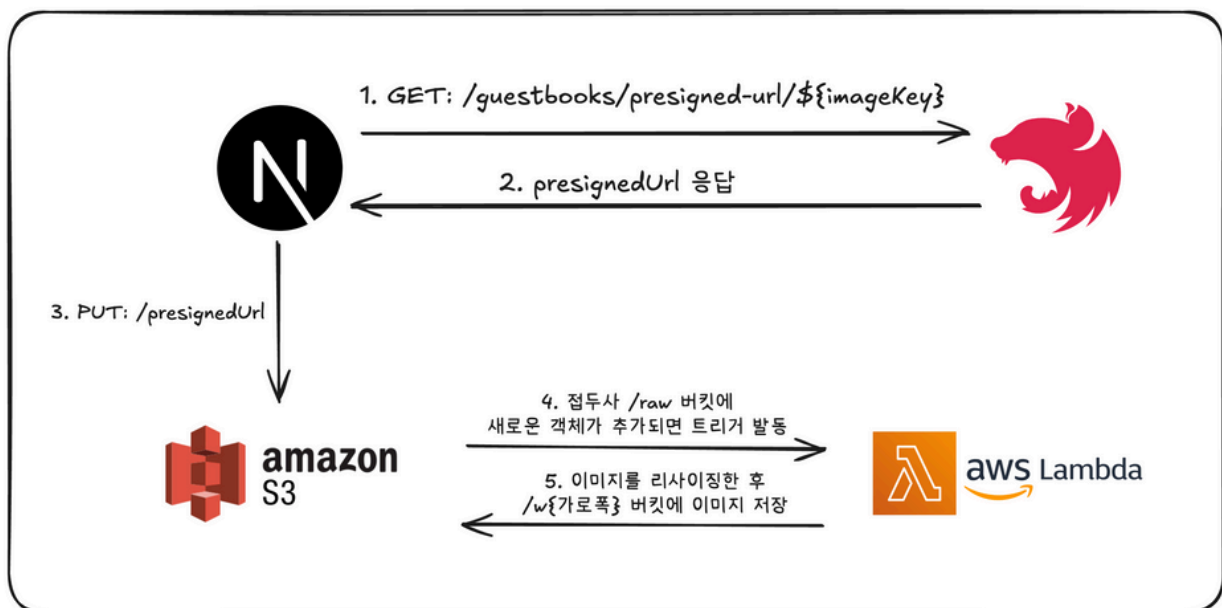
- Safari 포함 모든 주요 브라우저에서 이미지 필터 적용 및 저장 기능 정상화
- 캡처 방식이 아닌 원본 이미지를 저장하는 방식을 사용하여 이미지 품질 개선

문제: 대용량 이미지로 인한 페이지 로딩 성능 저하

- 상황: 필터 처리된 원본 이미지가 최적화 없이 직접 서빙됨
- 영향: 페이지 로딩 속도 저하로 사용자 경험 악화
- 세부사항: 불필요하게 큰 이미지 파일이 네트워크 대역폭 낭비

해결과정:

- 원인 분석
 - 페이지 로딩 병목 지점 파악(홍화면 무한 스크롤)
 - 이미지 사이즈 및 포맷 분석
- 해결 방안 설계
 - 이미지 최적화 전략 수립
 - AWS Lambda 기반 이미지 압축·저장 과정 자동화 설계
- 구현
 - AWS Lambda 이미지 처리 파이프라인 구축
 - 자동 리사이징 로직 구현: 원본(raw) → w140, w320, w640, w1024, w1440, w1920



결과:

- 이미지 파일 크기 98% 감소(640x960px 기준) -> 네트워크 대역폭 감소를 통한 페이지 렌더링 속도 향상
- 컴포넌트별 최적화된 이미지 제공

팀프로젝트 / 백엔드, DevOps

수준별 운동 매칭 서비스

2024.02 -2024.03

역할

- 사용자 랭킹 배치 프로세스
- GCP 클라우드 인프라 구축 및 서비스 배포/운영
- 검색 API
- 소셜 로그인 인증 시스템 구현 (OAuth 2.0)



랭크 시스템을 지원하는 수준별 운동 매칭 서비스

#NestJS #TypeScript #Prisma #PostgreSQL
#JWTTokenOAuth2.0 #Docker
#GCP(Cloud SQL, Cloud Storage, Artifact Registry, Cloud Build) #PostmanAPI

서버 

<https://github.com/jimyungkoh/sweatier-server>

문제와 해결

문제: 수동 빌드·배포의 비효율성

- 상황: 수동 배포로 인한 서비스 중단과 개발 지연
- 영향: 개발 생산성 저하, 서비스 불안정성
- 세부사항: 배포 과정의 휴먼 에러 위험

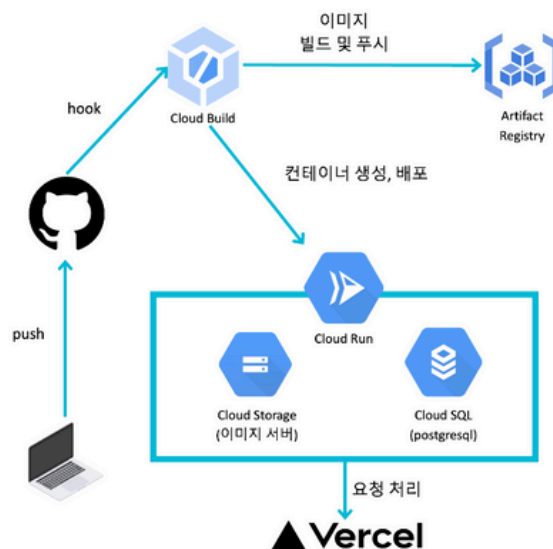
해결과정:

- 배포 자동화 설계
 - CI/CD 파이프라인 설계
 - 무중단 배포 전략 수립
 - 배포 검증 프로세스 정의
- 자동화 시스템 구축
 - CI/CD 파이프라인 구축
 - 배포 스크립트 개발
 - 서버 상태 검증 로직 구현

결과:

- 배포 프로세스 완전 자동화
- 서비스 중단 없는 배포 실현
- 배포 안정성 확보

프로덕션(production) 환경 아키텍처



문제: 예외 처리의 비효율성

- 상황: 도메인별로 분산된 에러 코드와 컨트롤러 중심의 예외 처리
- 영향: 코드 중복과 유지보수 어려움 발생
- 세부사항: 서버 측 에러 코드 중복, 컨트롤러 복잡성 증가

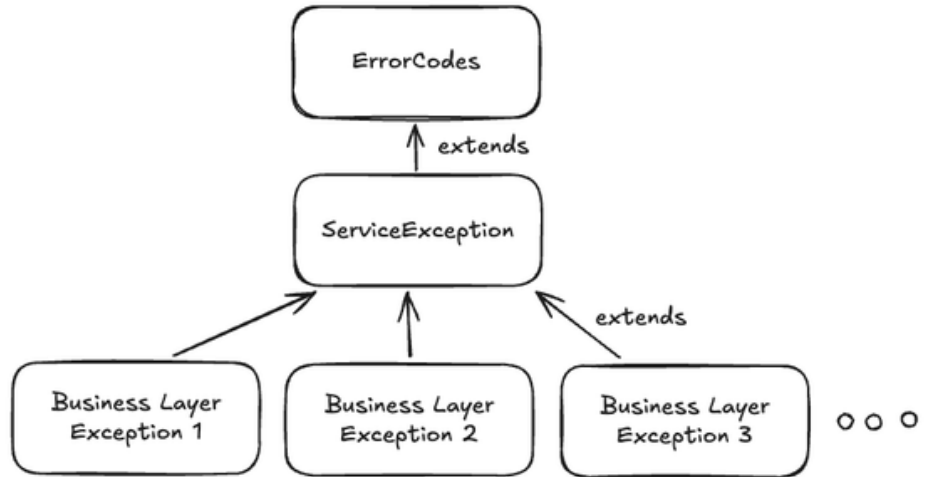
해결과정:

• 예외 처리 분석

- 현재 에러 코드 중복 현황 파악
- 컨트롤러의 예외 처리 패턴 분석
- 개선 방향 설계

• 중앙화된 예외 처리 구현

- 통합 에러 코드 시스템 설계
- Error 코드 상속 구조 구현
- 서비스 계층 예외 처리 로직 개발



결과:

- 에러 코드 중복 제거로 유지보수성 향상
- 컨트롤러 코드 복잡도 감소
- 일관된 예외 처리 체계 확보