대용량 트래픽 환경을 고려한 백엔드 시스템 설계 및 성능 검증 보고서

1. 프로젝트 개요

본 프로젝트는 이미지 및 PDF와 같은 대용량 파일을 데이터베이스에 직접 저장하지 않고 Object Storage에 저장하고 메타데이터만을 관리하는 백엔드 구조가 실제로 어떻게 동작하는지를 설계하고 구현 및 검증하는 것을 목표로 한다.

특히 특정 게시물 조회, 좋아요, 파일 업로드와 같이 트래픽이 집중되기 쉬운 기능을 중심으로 Spring Boot 기반 서버에서 Redis 캐시, 비동기 처리, Object Storage 연동을 적용한다.

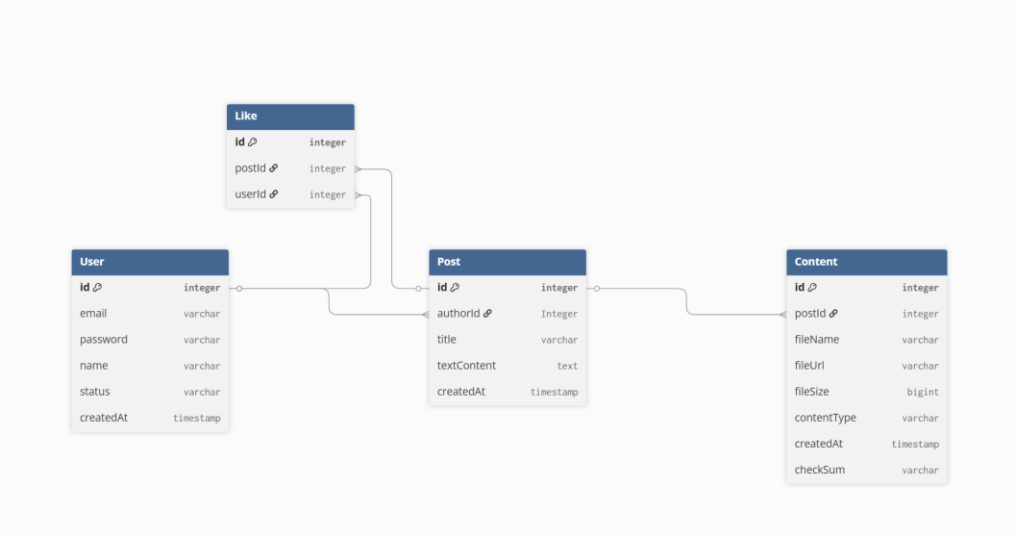
단순 기능 구현에 그치지 않고 실제 운영 환경을 가정한 대용량 트래픽 상황에서의 병목 지점과 한계를 명확히 정의하고 이에 대한 설계적 대응이 유요한가를 정량적으로 검증하는 데에 초점을 맞추어 진행하였다.

2. 시스템 설계

2-1. 도메인 및 스키마 설계

이미지 및 파일과 같은 대용량 데이터를 게시물 데이터와 분리하고 메타데이터 중심으로 조회 및 업로드, 다운로드가 이루어지는 구조를 목표로 한다. 특히 게시물 조회, 좋아요, 파일 업로드와 같이 트래픽이 집중될 수 있는 도메인에서 정합성 유지, 확장성, DB 보호를 우선 고려하여 설계하였다.

A. **Post 도메인**
   핵심 데이터만 관리하는 단순한 구조로, 이미지나 파일에 대한 정보를 포함하지 않고 게시물 제목과 본문 텍스트만 포함한다.
B. **Like 도메인**
   사용자와 게시물 간의 좋아요 관계를 관리하며 동시성 및 정합성 검증의 대상이다. MySQL과 Redis 카운터를 연동시켜 관리한다.
C. **User 도메인**
   인증 및 식별 목적으로 설계하였고 좋아요 및 업로드, 게시물 생성의 주체이다.
D. **Content 도메인**
   파일 자체는 Object Storage에 저장하며 DB에는 파일의 메타데이터만 저장한다.



2-2. API 설계

API는 게시물 조회, 좋아요, 파일 업로드와 같이 트래픽이 집중되기 쉬운 기능을 중심으로 설계되었다. 각 API는 동시성 발생 가능 지점과 부하 집중 구간을 명확히 구분할 수 있도록 구성하였으며 부하 테스트 및 동시성 검증 시나리오의 기준이 된다.

2-3. 실행 및 배포 환경 설계

로컬 및 테스트 환경에서 서비스 간 의존성을 명확히 분리하고 대용량 트래픽 및 동시성 테스트를 안정적으로 수행하기 위해 Docker Compose 기반으로 컨테이너 실행 환경을 구성하였다.

애플리케이션 서버, 데이터베이스, 캐시, Object Storage를 각각 독립된 컨테이너로 구성함으로써 각 컴포넌트의 역할을 분리하고 실제 운영 환경과 유사한 구조에서 테스트를 수행하였다.

(1) API 서버

스프링 부트 기반 백엔드 애플리케이션을 컨테이너로 실행한다. MySQL, Redis, MinIO 서비스에 의존하며 환경 변수는 도커 전용 ENV 파일로 분리하여 관리하였다.

(2) MySQL

게시물, 사용자, 파일의 메타데이터 등 핵심 메타데이터 저장소로 사용한다. Docker Volume을 통해 데이터 영속성을 보장하였고 healthcheck를 적용하여 DB가 정상 가동된 상태에서만 API 서버가 실행되도록 구성하였다.

(3) Redis

게시물 조회 성능 개선, 좋아요 수와 같이 트래픽이 집중되는 카운터성 데이

터를 처리하는 데 활용하였다. Redis는 조회 성능 개선을 위한 보조 수단으로 사용되며 데이터 정합성의 기준은 MySQL에 두도록 설계하였다.

(4) MinIO

이미지 및 PDF 파일과 같은 대용량 파일을 저장하기 위한 Object Storage로 사용하였다. 파일 자체는 MinIO에 저장하고 DB에는 파일의 식별자와 메타데이터만 저장하는 구조로 설계하였고 MinIO 또한 Docker Volume을 통해 데이터 영속성을 보장하였다.

2-4. 테스트 환경 설계

기능 구현의 정확성과 동시성 및 트래픽 환경에서의 안정성을 검증하기 위해 단위 테스트, 통합 테스트, 동시성 테스트, 부하테스트를 단계적으로 수행하였다. 테스트 환경 또한 application-test.yml으로 분리하여 구성하였고 기본 DB는 H2 인메모리 데이터베이스를 사용하여 테스트 간 독립성과 반복 실행 가능성을 확보하였다.

(1) 단위 테스트 및 통합 테스트

JUnit 기반으로 각 도메인의 Service 계층과 Controller 계층 단위테스트, 통합 테스트를 작성하였다. 이를 통해 실제 데이터베이스 환경과 유사한 흐름을 검증하면서도 테스트 실행 비용을 최소화하였다.

(2) 동시성 테스트

좋아요 기능과 같이 동시 요청이 집중될 수 있는 로직을 대상으로 JUnit 기반으로 동시성 테스트를 수행하였다. 동시성 테스트는 실제 MySQL 환경에서 수행하였으며 테스트 전용 데이터베이스를 분리하여 사용하였다.

(3) 부하 테스트

대용량 트래픽 환경에서의 응답 성능과 안정성을 검증하기 위해 k6를 사용한 부하 테스트를 수행하였다. 부하 테스트는 게시물 조회, 좋아요 요청 등 트래픽 집중 가능성이 높은 API를 중심으로 진행하였다. 부하 테스트 환경에서는 MySQL을 실제 데이터베이스로 사용하여 캐시 및 DB 부하 분산 효과를 검증하였다.

3. 트래픽 및 동시성 위험 분석

본 프로젝트에서는 다음과 같은 상황을 대용량 트래픽 환경에서 발생 가능한 핵심 문제로 정의하였다.

3-1. 조회 트래픽 집중으로 인한 DB 병목

게시물 조회 요청이 특정 콘텐츠에 집중될 경우 동일한 조회 쿼리가 반복적

으로 DB에 전달되어 병목이 발생할 수 있다. 이는 응답 지연 증가 및 DB 부하 누적으로 이어질 가능성이 있다.

3-2. 좋아요 기능에서의 동시성 및 정합성 문제

다수의 사용자가 동시에 좋아요 및 좋아요 취소를 요청할 경우 경쟁 조건으로 인해 좋아요 수 불일치가 발생할 수 있다. 캐시와 DB를 함께 사용하는 본 프로젝트의 구조에서는 카운터 값의 정합성 보장 여부가 핵심 문제가 된다.

3-3. 파일 업로드 요청 집중 시 서버 부하

이미지 및 PDF 파일을 포함한 업로드 요청이 동시에 유입될 경우 서버 자원 소모 및 응답 지연이 발생할 수 있으며 파일 저장 로직이 DB와 강하게 결합된 경우 전체 서비스 안정성에 영향을 줄 수 있다.

3-4. 불필요한 DB 접근으로 인한 보호 실패

캐시 미적용 또는 동기 처리 구조에서는 트래픽 증가시 DB가 직접적인 부하를 받게 되며 이는 전체 시스템의 확장성과 안정성을 저해하는 요소가 된다.

4. 테스트 설계

본 프로젝트는 위에서 정의한 문제들이 설계 변경을 통해 실제로 완화되거나 해결되는지를 아래 항목을 중심으로 검증하였다.

A. Redis 캐시 적용 전 / 후의 게시물 조회 성능 비교
B. 동시 좋아요 및 좋아요 취소 요청에서의 데이터 정합성 유지 여부
C. Redis 카운터와 DB 간 좋아요 수 일관성 검증
D. 비동기 이벤트 처리 적용 시 DB 직접 부하 감소 효과
E. Object Storage 기반 파일 업로드 구조에서의 동시성 처리 안정성

각 검증은 테스트 성공 여부 뿐만 아니라 응답 시간, p95 / p99 지연, 성공률, 데이터 일관성을 기준으로 분석하였다.

5. 테스트 시나리오

A. 게시물 조회 부하테스트

게시물 조회 API는 트래픽이 특정 리소스에 집중되기 쉬운 대표적인 읽기 요청이다. 따라서 이 테스트는 Redis 캐시 적용 전후의 성능 차이 비교, DB 직접 조회 구조에서 발생하는 병목 여부 확인을 목적으로 테스트를 진행하였다.

성능 지표	Redis 적용 전 (Baseline)	Redis 적용 후	개선율 (%)
평균 응답 (Avg)	629 ms	491 ms	▼ 21.9%
중간값 (Med)	580 ms	492 ms	▼ 15.2%
Tail Latency (p95)	1,690 ms	1,060 ms	▼ 37.3%
최대 응답 (Max)	2,890 ms	2,200 ms	▼ 23.9%

평균 응답 시간이 약 22% 감소하였고 p95 지연 시간 또한 약 37% 개선하였다. 또한 동일 조건에서 처리 가능한 동시 사용자 수가 약 2.7배 증가하였다. 이는 게시물 조회 요청의 상당수가 DB를 거치지 않고 Redis 캐시에서 처리되었음을 의미하며 DB 병목이 완화되었음을 수치로 확인할 수 있었다. Redis 캐시는 단순히 평균 응답 시간을 줄이는 역할을 넘어 일부 요청이 장시간 대기하는 Tail Latency 구간을 효과적으로 완화하였다. 이를 통해 사용자 체감 성능과 SLA 관점에서 유의미한 개선이 이루어졌음을 확인하였다.

B. 좋아요 동시성 및 정합성 테스트

좋아요 기능은 동일 게시물과 사용자에 대해 동시 쓰기 요청이 발생하는 대표적인 동시성 케이스이다. 이 테스트의 목적은 동일 사용자 – 게시물 조합에 대해 중복 좋아요가 생성되지 않는지, 동시 요청 환경에서도 Like 데이터가 1건만 생성되는지를 확인하는 것이다.

테스트 결과 동시 요청 상황에서도 Like 테이블에는 1건의 row만 생성하였고 중복 데이터 저장, 데이터 손실 등 비정상적인 상황이 발생하지 않았고 5번의 반복 테스트에서도 동일한 결과를 확인하였다.

C. 좋아요 DB 보호 및 비동기 처리 검증

동시에 좋아요 요청이 집중될 경우 Deadlock, Unique Key Violation과 같은 DB 레벨의 문제가 발생할 수 있다. 이 테스트에서는 DB 충돌 상황에서도 시스템이 안정적으로 동작하는지를 검증하는 것이다.

동일한 게시물과 사용자에 대해 다수의 스레드가 동시에 좋아요 요청을 하면 DB 레벨에서 Unique Constraint가 적용이 되고 충돌 발생 시 재시도 로직의 동작 여부를 검증하는 시나리오로 테스트를 진행하였다.

그 결과 동시 요청 상황에서도 좋아요 테이블에서는 항상 1건의 데이터만을 유지하였고 Deadlock 또는 Unique Key Violation으로 인한 시스템 중단이 발생하지 않았다. 총 5번의 반복 실행 환경에서도 테스트 전부 통과하였다.

이는 애플리케이션 레벨의 보호 로직과 DB 레벨의 제약 조건이 함께 작동하는 구조가 DB 보호에 효과적임을 보여준다.

D. 업로드 동시성 테스트

대용량 파일 업로드는 네트워크 전송, Object Storage I/O, 비동기 처리 비용이 동시에 발생하는 작업이다. 이 테스트는 다수의 사용자가 동시에 파일을 업로드하는 상황에서 시스템의 안정성을 검증하고 업로드 요청이 서버를 블로킹하지 않고 처리되는지를 확인하였다.

K6를 이용하여 테스트를 진행하였고 파일은 약 2.4MB 크기로 진행하였다. 첫 테스트에서는 30초가 100명의 가상 사용자의 요청을 보냈고 95%의 요청이 1초안에 완료되도록 설계하였으나 p95의 시간 초과로 테스트가 실패하였다. 그러나 각 요청의 실패율은 0%, 서버 다운은 발생하지 않았다.

이는 업로드 처리 특성상 응답 지연이 발생할 수 있음을 고려하지 않은 과도하게 엄격한 SLA 기준 설정의 한계로 판단하였다.

이후 테스트에서는 점진적으로 트래픽을 증가시키도록 설정하였고 95%의 요청이 10초 이내에 완료되도록 설계하니 실패율 0%, 서버 안정성을 유지할 수 있었다.

이를 통해 업로드 처리에서는 응답 속도보다 시스템의 안정성과 실패율 관리가 더 중요한 지표임을 확인하였다.

6. 종합 분석 및 설계 평가

본 프로젝트는 메타데이터 중심의 백엔드 구조를 기반으로 대용량 트래픽 및 동시성 환경에서 발생할 수 있는 주요 문제를 정의하고 설계 변경을 통해 해당 문제가 실제로 완화되거나 해결되는지를 검증하는 것을 목표로 하였다.

[조회 트래픽 처리 구조의 개선]

Redis 캐시를 게시물 조회 API에 적용함으로써 DB 직접 조회 구조에서 발생하던 병목을 효과적으로 완화할 수 있었다. 테스트 결과 평균 응답 시간뿐 아니라 p95 지연 시간이 크게 개선되었으며 이는 일부 요청이 장시간 대기하는 Tail Latency 구간이 완화되었음을 의미한다. 이를 통해 캐시 계층이 단순 성능 개선을 넘어 사용자 체감 성능과 SLA 안정성 측면에서도 유의미한 역할을 수행함을 확인하였다.

[동시성 환경에서의 데이터 정합성 확보]

좋아요 기능을 대상으로 수행한 동시성 테스트를 통해 다수의 동시 요청 환경에서도 데이터 정합성이 유지됨을 검증하였다. 애플리케이션 레벨의 보호 로직과 DB 레벨의 유니크 제약 조건을 함께 적용함으로써 중복 데이터 생성, 데이터 손실, DB 충돌로 인한 서비스 중단과 같은 문제를 방지할 수 있었다. 이는 동시성 제어를 특정 계층에만 의존하지 않고 여러 계층에서 중복 방어하도록 설계한 접근이 효과적이었음을 보여준다

[DB 보호 관점에서의 비동기 처리 효과]

좋아요 요청 처리 과정에서 비동기 이벤트 기반 구조를 적용함으로써 DB에 직접 집중되는 부하를 완화할 수 있었다. 특히 동시 요청이 집중되는 상황에서도 DB 충돌이 서비스 장애로 전파되지 않았으며 반복 테스트 환경에서도 안정적인 동작을 유지하였다. 이를 통해 비동기 처리는 성능 최적화뿐 아니라 시스템 안정성을 확보하기 위한 설계 요소임을 확인하였다.

[대용량 파일 업로드 처리의 안정성 검증]

Object Storage를 활용하여 파일 데이터를 DB에서 분리하고 메타데이터만을 관리하는 구조를 통해 대용량 파일 업로드 환경에서도 시스템 안정성을 확보할 수 있었다. 초기 테스트에서는 과도하게 엄격한 SLA 기준으로 인해 테스트가 실패로 판정되었으나, 실패율 0%와 서버 무중단 상태를 바탕으로 기준 설정 자체의 한계를 식별할 수 있었다. 이후 업로드 도메인의 특성을 고려한 현실적인 기준을 적용함으로써 실제 운영 환경에서 의미 있는 안정성 지표를 도출할 수 있었으며, 대용량 파일 처리에서는 응답 속도보다 시스템 안정성과 실패율 관리가 더 중요한 지표임을 확인하였다.

[종합 평가]

본 프로젝트를 통해 메타데이터 중심의 백엔드 구조와 Redis 캐시, 비동기 처리, Object Storage 분리 설계가 대용량 트래픽 및 동시성 환경에서 실질적인 효과를 가진다는 점을 정량적인 테스트를 통해 검증하였다. 특히 단순 구현이 아닌 문제 정의 → 설계 변경 → 검증 → 해석의 과정을 반복함으로써 설계 판단이 실제 시스템 동작에 어떤 영향을 미치는지를 확인할 수 있었다. 이는 향후 실제 서비스 환경에서 유사한 트래픽 및 동시성 문제에 대해 합리적인 설계 결정을 내리는 데 기초 자료로 활용될 수 있을 것이다.