

황준하

숫자로 증명하는 백엔드 개발자

h_leopold@naver.com

github.com/backendVirtuoso

hjh-portfolio.vercel.app

자기소개

"측정할 수 없는 개선은 확신할 수 없기에, 데이터로 증명합니다."

기능 구현을 넘어 시스템의 한계를 집요하게 파고들기를 즐깁니다. DB 커넥션 고갈을 유발하는 병목 지점을 진단하고 **Redis Cache-Aside 전략**으로 돌파했습니다. **조회 지연 시간을 1/12(60ms→5ms)로 단축**하고 쿼리 부하를 차단한 경험이 있습니다.

"막연한 추측 대신, 확실한 구조로 안정성을 확보합니다."

기술 선택의 트레이드 오프를 고려합니다. Kafka 기반 Event-Driven 아키텍처를 설계하여 I/O 부하를 분산하고 Non-blocking 시스템을 구축한 경험이 있습니다. 읽기와 쓰기의 물리적 책임을 분리함으로써, 트래픽 스파이크 상황에서도 메시징 유실 없는 무중단 서비스 토대를 완성했습니다.

기술 스택

Java

Spring Boot

Next.js

FastAPI

Python

JPA

Kafka

MySQL

Typescript

경력

(주)포테이토넷 일 경험 인턴 2025.10.13 - 2026.01.02

[프로젝트 1: CypherThreat - 웹 보안 위협 분석 및 프로파일링 플랫폼]

- 개요:** JS 코드 분석 및 그래프 데이터베이스를 활용하여 악성 URL 캠페인을 식별하고 추적하는 보안 분석 시스템
- 기술 스택:** FastAPI, MongoDB, MySQL, TypeScript, Next.js

[대용량 스크립트 데이터 처리를 위한 데이터베이스 아키텍처 최적화]

문제

MongoDB에 저장된 대용량 스크립트 데이터 조회 시 인덱스 미비와 중복 요청으로 인한 서버 부하가 발생

해결

- url_id 기반의 복합 인덱스 설정 및 Projection 적용으로 쿼리 실행 계획을 최적화.
- 애플리케이션 레벨에서 **LRU 캐싱 전략(TTL 5분)**을 도입하여 반복적인 DB 접근을 최소화.
- BSON 데이터 전송 시 zlib 압축을 적용하여 네트워크 트래픽을 효율화.

결과

대규모 트래픽 상황에서도 빠른 실패(Fail-fast) 처리와 안정적인 데이터 조회가 가능한 백엔드 환경을 구축

[프로젝트 2: Olpemi Web - 캠페인 관리 시스템]

- **개요:** JS 클러스터링 기반 피싱 캠페인 통계 및 관리를 위한 어드민 시스템
- **기술 스택:** Next.js, TypeScript, Material-UI, Recoil

[스크린샷 탭 API 호출 병렬 처리 최적화]

문제

문제 순차적 API 호출로 인해 20개 스크린샷 해시 로딩에 2.25초 소요, Decimal 해시 값의 JavaScript 정밀도 손실 발생

해결

- for 루프를 **Promise.all()**로 변경하여 모든 API를 병렬 호출.
- BigInt 기반 Decimal→Hex 변환을 구현하여 정밀도 문제 해결.
- 개별 try-catch로 에러 발생 시에도 나머지 결과가 정상 반환되도록 처리.

결과

로딩 시간 90% 단축(2.25초 → 0.2~0.4초), 해시 변환 정확도 100% 유지

1. K-Sketch - 실시간 채팅 및 여행 경로 플랫폼

- 2024.10 ~2025.03 (5개월) | Backend Developer (5인 팀)
- Link : github.com/backendVirtuoso/K-Sketch

"RDB에 대한 강한 의존성이 확장성의 발목을 잡고 있었습니다."

문제: 데이터 중복 호출과 병목 현상

채팅방 입장 시마다 동일한 메시지 내역을 DB에서 반복 조회하며 발생하는 과도한 Disk I/O가, 커넥션 풀 고갈의 직접적인 원인이었고 트래픽 증가 시 서비스 전체 마비로 이어질 수 있는 구조적 결함을 해결하기 위해, DB 접근을 원천 차단하는 아키텍처로 재설계했습니다.

[해결 전략 1: 조회 성능 가속화]

문제

변하지 않는 과거 대화 내역에 대해 반복적으로 발생하는 **Disk I/O가 시스템 자원을 낭비함**.

해결

채팅방별 '최근 메시지 Top 100'을 **Redis(In-Memory)**에 캐싱하여 DB 접근을 원천 차단하는 구조 도입.

전략

Look Aside 패턴을 적용하여 **Cache Miss**가 발생할 때만 DB를 참조하도록 설계했습니다.

이를 통해 다수의 유저가 동시에 입장하더라도 **DB 부하 없이 메모리에서 데이터를 즉각 반환**하도록 최적화했습니다.

[해결 전략 2: 쓰기 부하 분산]

문제

채팅 메시지 저장(Insert) 트래픽이 몰릴 경우, **동기적인 DB 처리가 메인 서버 스레드를 블로킹할 위험** 확인.

해결

Kafka를 메시지 브로커로 도입하여 **채팅 전송 로직과 DB 저장 로직을 물리적으로 분리**.

전략

비동기 이벤트 처리(Event-Driven) 방식을 적용했습니다. 클라이언트 메시지를 Kafka Topic으로 발행하고, Consumer가 이를 구독하여 DB에 저장함으로써, 순간적인 트래픽 스파이크(Spike) 상황에서도 **메시지 유실 없이 안정적인 처리가 가능한 느슨한 결합(Loose Coupling)** 시스템을 구축했습니다.

검증

DB로 집중되던 I/O가 Redis(조회)와 Kafka(쓰기)로 분산됨에 따라, 메인 비즈니스 로직이 DB 응답 지연에 영향받지 않는 **비차단(Non-blocking)** 구조가 완성됨을 확인했습니다. 특히 메시지 브로커를 둬으로써, 추후 트래픽 증가 시 Consumer 인스턴스만 늘리면(Scale-out) 처리량을 유연하게 조절할 수 있는 **확장 가능한 토대**를 성공적으로 마련했습니다.

회고

캐시와 메시지 큐 도입으로 시스템 복잡도는 증가했지만, 이 경험을 통해 '오버엔지니어링 경계'와 '적정 기술 선택'의 중요성을 배웠습니다. 비록 대규모 실트래픽을 경험하진 못했지만, "지금 짠 코드가 100만 명의 요청을 받는다면?"이라는 가정하에 설계를 고민하는 과정에서, 단순히 기능을 만드는 코더에서 시스템을 설계하는 엔지니어의 시야를 갖게 된 중요한 전환점이었습니다.

2. Jeju-Suffer - 제주 해수욕장 정보 및 서핑 예약 서비스

- 2024.06.21 ~2024.07.03 (2주일) | Backend Developer (6인 팀)
- Link : github.com/backendVirtuoso/gsitm-spring-environment-monitoring

"외부 API의 복잡한 응답 구조를 분석하고, 안정적인 데이터 파싱 로직을 설계합니다."

문제

외부 날씨 API 연동 시 복잡한 JSON 중첩 구조로 인한 파싱 오류 및 대량 게시글 조회 시 성능 저하 발생.

해결

응답 전문 분석을 통해 DTO 매핑 로직을 재설계하여 예외 처리를 강화하고, JPA Pageable을 적용해 쿼리를 최적화.

검증

데이터 파싱 오류를 해결하여 서비스 안정성을 확보하고, 대량 조회 시에도 일정한 응답 속도를 유지하는 효율적인 조회 환경 구축.

3. 개인 포트폴리오 웹 사이트 구축 - Next.js

- 2025.11.04 ~2025.11.23 (3주) | Backend Developer (개인)
- Link: github.com/backendVirtuoso/portfolio | hjh-portfolio.vercel.app

"복합 필터링과 SPA 구조로 데이터 탐색 효율을 극대화한 웹 서비스"

문제

기존 Spring Boot로 배포하였던 프로젝트는 비용 문제와 새로 학습한 Next.js를 이용하고자 함

해결

Next.js v15 App Router의 서버 컴포넌트, SSG로 초기 로딩 최적화, GitHub와 Vercel 연동으로 배포 자동화

검증

빌드 및 배포 과정을 자동화하여 배포 소요 시간을 기존 5분에서 40초 내외로 86.6% 이상 단축하고 운영 효율성 극대화.

4. 경기도 사고 유형별 위치 조회 프로젝트

- 2024.05.28 ~2024.06.04 (1주) | Backend Developer (6인 팀)
- Link: github.com/backendVirtuoso/gsitm-react-traffic-safety | <https://backendvirtuoso.github.io/gsitm-react-traffic-safety/>

"GitHub Actions 기반 자동 배포 파이프라인 구축으로 협업 워크플로우 개선"

문제

방대한 공공 데이터 내에서 사용자가 원하는 특정 조건의 사고 정보를 효율적으로 탐색하고 관리하는 데 어려움 존재.

해결

사고유형/지역/연도 등 복합 조건 필터링 로직을 구현하고, React Router를 활용해 페이지 로딩 없는 SPA 방식으로 설계.

검증

데이터 탐색 시간을 단축하고 즉각적인 화면 전환을 제공하여 사용자 흐름(User Flow)에 최적화된 UX 개선 달성.

5. 개인 포트폴리오 웹 사이트 구축 - Spring Boot

- 2024.07.29 ~2024.08.18 (1개월) | Backend Developer (개인)
- Link: github.com/backendVirtuoso/portfolio-website

“API·도메인 구조를 직접 설계하며 백엔드 아키텍처 기초를 다진 프로젝트”

문제

코드 수정 시마다 수동으로 빌드/배포하는 과정에서 평균 10분 이상의 시간이 소요되어 개발 생산성이 저하

해결

GitHub Actions로 main 브랜치 푸시 시 자동 빌드 - Cloudtype 배포 워크플로우 구현

검증

빌드 및 배포 과정을 자동화하여 배포 소요 시간을 기존 10분에서 2분 내외로 80% 이상 단축하고 운영 효율성 극대화.

교육/자격증

[GSITM] 부트캠프 3기 (2024.03.27 - 2024.07.25)

- 무질서했던 협업 환경을 개선하기 위해 **Git Flow 기반의 브랜치 전략**을 설계·표준화하여, 무분별한 커밋으로 인한 Merge Conflict 문제를 근본적으로 해결하고 **충돌 발생률 0건을 유지**했습니다.
- 설계 주도의 개발 문화 정착을 위해 **팀 내 코드 컨벤션 가이드라인 수립** 및 공통 로직 모듈화를 주도하여, 코드 중복률을 낮추고 유지보수 생산성을 높였습니다.

자격증 및 학력

- 자격증 : 정보처리기사 (2024.06), SQL개발자(SQLD) (2024.06)
- 학력 : 경기과학기술대학교 컴퓨터모바일융합공학과 전공심화과정 학사졸업 (4.11 / 4.5)
경기과학기술대학교 컴퓨터모바일융합과 전문학사졸업 (4.0 / 4.5)

자기소개서

지원동기

[왜 백엔드 개발자인가]

시스템의 병목을 진단하고 해결하는 백엔드 개발자

저는 사용자가 체감하는 '느림'의 원인을 끝까지 추적하고, 구조적으로 해결하는 개발자가 되고자 합니다. K-Sketch 프로젝트에서 **Kafka 기반 실시간 채팅 서비스**를 구현했습니다. 기능은 정상 작동했지만, 동시 접속자가 늘어날수록 메시지 조회 속도가 눈에 띄게 느려졌습니다. 원인을 분석한 결과, 채팅방 입장 시마다 MySQL에서 동일한 메시지를 반복 조회하며 발생하는 Disk I/O가 커넥션 풀을 고갈시키고 있었습니다. **Redis Cache-Aside 전략을 도입**하여 '최근 메시지 Top 100'을 메모리에 캐싱했고, 조회 지연 시간을 **60ms에서 5ms로 단축**했습니다. 이 경험을 통해 백엔드 개발자의 핵심 역량은 단순 기능 구현이 아니라 **"왜 느린가?"를 파고들어 구조적으로 해결하는 것**임을 깨달았습니다. 입사 후에는 서비스의 병목 지점을 선제적으로 진단하고, 시스템 전체 관점에서 최적의 해결책을 제시하는 개발자로 기여하겠습니다.

성격의 장단점

[장점: 끝까지 파고드는 집요함]

"JSON 파싱 오류, 3일간의 삽질이 남긴 것"

Jeju-Suffer 프로젝트에서 외부 날씨 API를 연동하는 과정에서 JSON 파싱 오류가 발생했습니다. 단순히 예외 처리로 넘어갈 수 있었지만, "왜 포맷이 다른가?"를 3일간 파고들었습니다. API 문서를 다시 분석하고, 실제 응답 구조와의 차이점을 정리한 뒤, 다양한 응답 형태에 대응할 수 있는 유연한 DTO 매핑 로직을 설계했습니다. 이 과정에서 **임시방편이 아닌 근본 해결**이 결국 팀의 시간을 아낀다는 것을 배웠습니다. 입사 후에도 문제를 발견하면 원인까지 파고들어, **같은 이슈가 반복되지 않는 시스템**을 만들겠습니다.

[단점 및 개선: 소통의 중요성을 배운 경험]

"기획 없이 시작한 프로젝트, 2주간의 혼란"

K-Sketch 프로젝트 초기, 팀원 간 충분한 논의 없이 각자의 해석으로 개발을 시작했습니다. 결과는 예상대로였습니다. 기능 간 충돌, API 스펙 불일치, 2주간의 작업 지연. 문제를 인식한 후 정기 회의를 제안했습니다. API 스펙 문서화, 역할 분담 재조정, 매일 15분 스탠드업 미팅을 도입했습니다. 이후 충돌 발생 건수가 눈에 띄게 줄었고, 프로젝트를 성공적으로 완성했습니다. 개발자에게 기술력만큼 중요한 것은 **"함께 만들어가는 구조"**를 설계하는 능력입니다. 지금은 프로젝트 시작 전 기획과 역할 분담을 먼저 확인하고, 적극적으로 방향을 조율하는 습관을 갖추게 되었습니다.

직무 관련 경험 및 역량

[실전에서 검증한 백엔드 역량]

"Redis 캐싱 도입으로 조회 속도 12배 개선"

K-Sketch 프로젝트에서 채팅방 입장 시 발생하는 반복적인 DB 조회가 시스템 병목의 원인임을 진단했습니다. Redis Cache-Aside 패턴을 적용하여 '최근 메시지 Top 100'을 캐싱했고, 조회 지연 시간을 1/12로 단축(60ms→5ms) 했습니다. 이를 통해 다수의 유저가 동시 입장해도 DB 부하 없이 메모리에서 데이터를 즉각 반환하는 구조를 구축했습니다.

"쓰기 부하 분산 - Kafka Event-Driven 아키텍처"

메시지 저장 트래픽이 몰릴 경우 동기적 DB 처리가 메인 서버 스레드를 블로킹할 위험을 확인했습니다. **Kafka**를 메시지 브로커로 도입하여 **채팅 전송 로직과 DB 저장 로직을 물리적으로 분리**했습니다. 비동기 이벤트 처리 방식을 적용하여 트래픽 스파이크 상황에서도 **메시지 지 유실 없이 안정적인 처리**가 가능한 느슨한 결합(Loose Coupling) 시스템을 구축했습니다.

"CI/CD 자동화 - 배포 효율성 극대화"

개인 포트폴리오 프로젝트에서 코드 수정 시마다 **수동 빌드/배포에 평균 10분**이 소요되는 문제를 해결하기 위해 **GitHub Actions 기반 자동 배포 파이프라인**을 구축했습니다. main 브랜치 푸시 시 자동 빌드-Cloudtype 배포 워크플로우를 구현하여 **배포 소요 시간을 10분에서 2분으로 80% 단축**했습니다.

입사 후 포부

[빠르게 적응하고, 실질적으로 기여하는 개발자]

"3개월: 시스템 파악, 6개월: 기능 개선 제안"

입사 직후에는 회사의 백엔드 아키텍처, API 설계 방식, 배포 파이프라인, CI/CD 환경을 집중적으로 학습하겠습니다.

3개월 내 코드 구조와 시스템 흐름을 파악하고, 유지보수 및 버그 수정에 실질적으로 기여하겠습니다.

6개월 후에는 기존 기능의 개선 방향을 주도적으로 제안하거나, 신규 기능 아이디어를 프로토타입으로 구현해 기술적으로 기여하겠습니다.

코드 품질과 아키텍처 개선에 기여하며, 단순히 개인의 성장을 넘어 팀 전체의 기술 수준을 높이는 구성원이 되는 것이 목표입니다.