구동현

Ku Donghyun

Back-End Developer

(클릭)

: 1999.01.21 : 010-9220- 3795 : kudongku@naver.com

객체지향을 중시하고, 성능과 효율을 위해 리팩토링을 거듭한 경험이 있는 개발자입니다.
페어 프로그래밍, 기술 블로그, 코드 리뷰 등 다양한 방식의 협업을 좋아합니다.
TDD 중심의 프로젝트에서 코드 커버리지 100%를 이끌었습니다.

STACKS.

Language	Java
Framework & Library	Springboot, Junit5, SpringSecurity, JWT
DB & Server	MySQL, QueryDSL, Docker, Redis, GithubActions
AWS	ECS, ECR, EC2, CloudWatch, ElastiCache, Aurora, VPC, ASG, CloudFront

CERTIFICATE. (클릭)

2024.06
- AWS Solution Architect Associate

2023.02
- TOEIC - 935점

2018.03 ~ 2024.08
- 건국대학교 융합생명공학과 학사 졸업

EDUCATION.

2024.08 ~ 2024.09
- (주) 소울웨어 인턴 근무

2023.12 ~ 2024.05
- 내일배움캠프 스프링 4기 수강
- 내향형 프로토콜 팀장으로 땅땅땅이라는 프로젝트를 기획

2023.10 ~ 2023.11
- PROJECT-X : PBL방식으로 시작하는 파이썬, 자바 수강
- 42SEOUL과 건국대학교에서 주최한 한달 단기 프로젝트
- 게시판 서비스를 구현하며 자바 1등을 수상

PROJECT EXPERIENCE.

땅땅땅

- 링크 : 주요기능 시연영상 Github 발표영상 (클릭)

2024.03.26 - 2024.05.02 (5주), 팀 프로젝트 (BE : 4명), Team Leader 포지션

사용자의 위치 정보를 기반으로 근처에 사는 이웃들과 함께하는 실시간 경매 서비스

Front End	React, Axios
Back End	Java, Springboot, SpringSecurity, JWT, Junit5
Collaboration	Slack, Notion, Github
Infrastructure	Docker, GithubActions, AWS(ECS, ECR, EC2, CloudWatch, ElastiCache, Aurora, VPC, ASG)

담당업무 :

Back End
- Redis Keyspace Notification을 통한 경매 상태 자동 변경을 통한 경매의 실시간성 반영
- S3 이미지 업로드 기능 구현
- QueryDSL의 Projection을 통해 Repository에서 Entity 대신 DTO를 받는 JPA 최적화
- TDD 중심의 개발로 코드 커버리지 100% 달성
- 유저 피드백을 반영해서, Redis 캐싱 적용한 경매글 조회 성능 개선 (처리량 10.3TPS -> 54.9TPS, 약 4.3배 향상)

DevOps
- 깃허브 액션 + 도커이미지 + ECR + ECS로 이어지는 CICD 파이프라인 구축
- CloudFront로 S3 이미지 캐싱해 성능개선 (응답속도 54.45ms -> 10.67ms, 약 80% 개선)

ETC
- React, Axios를 사용한 메인 페이지, 회원가입/로그인 페이지, 상세 페이지 구현
- 카카오 API를 통해 지도 핀 서비스 구현
- 팀리더로서 Ground Rules, Code Convention, Commit Rule 세팅

트러블 슈팅 및 리팩토링 :

1. JPA에서 Entity 대신 DTO로 조회를 통한 JPA 최적화 자세히

[Before]
- Repository에서 직접 entity를 받고, entity를 통해 DTO를 생성
- 응답 속도 : 355ms

[After]
- QueryDSL의 Projection을 사용하여, Entity의 모든 컬럼을 조회하는 대신 필요한 컬럼만 조회
- 응답 속도 : 52ms (약 6.5배 개선)

2. 유저테스트 피드백 후, 캐싱 성능 개선 자세히

[1단계] 경매글 전체 조회 메인 로직 구현
- QueryDSL을 사용해 메인 로직인 유저의 인근 지역 리스트와 필터링 조건을 통해 Repository에 조회
- 유저테스트 결과 경매글 조회 로직이 느리다는 피드백을 받았고, 회의 후 캐싱을 적용하기로 결정
- 처리량 : 10.3TPS

[2단계] Ehcache 적용
- JAVA 기반 오픈 소스 라이브러리인 Ehcache를 적용하기로 결정
- Redis와 달리 별도의 데몬을 거치지 않고, 로컬에서 돌아가서 성능적인 장점이 있었음.
- 처리량 : 72.6TPS (기존 대비 / 약 6배 향상)

[3단계] 데이터 부정합성에 대한 고민 후, Redis 캐싱으로 최종 변경
- Ehcache가 로컬 캐시라서 현재 서버가 2개 이상 있기에 데이터 부정합성 발생
- 트레이드 오프를 고민한 뒤, Redis Cache로 2차 변경
- Redis를 pub/sub, keyspace, 분산락을 위해 이미 사용중이라서 인프라 구축 또한 필요없었음
- 처리량 : 54.9TPS (기존 대비 / 약 4.3배 향상)

3. S3 이미지에 CloudFront(CDN) 적용 후, 성능 개선 자세히

[Before]
- 기존에는 S3 엔드포인트로 경매글 이미지를 조회할 수 있었음.
- 비용적인 측면이나 성능적인 측면에서 S3에서 이미지를 직접 조회하지 않는 방안 탐색
- 응답 속도 : 54.45ms

[After]
- CloudFront의 엣지포인트에 캐싱해놓은 데이터를 통해 S3 버킷에 대한 **자체적인 부하**를 개선
- 응답 속도 : 10.67ms (약 80.4% 개선)

기술적 의사결정 :

1. 메인 API 서버의 부하를 줄이기 위한 SSE 서버 분리

[상황 및 고려 사항]
- 대규모 트래픽을 염두해두고 만든 서비스인 상황
- 메인 API 서버에 SSE를 구현했을 경우, Coupling에 대한 우려
- SSE로 인해 오토스케일링시 서버의 규모를 크게 잡기 때문에 서버 리소스가 낭비될 것을 우려

[결정]
- 메인 API 서버와 SSE 서버를 분리

2. 개발 전, 무중단 자동화 배포를 위한 ECR -> ECS Pipeline 구축

[상황 및 고려 사항]
- EC2 VS Fargate(container)를 우선 고려, 땅땅땅이 초기 서비스인 점을 감안
- EC2를 사용할 시, 직접 관리할 인적 리소스가 필요

[결정]
- Fargate에서 작업할 시, 실행 중인 작업 수에 따라 리소스가 자동으로 할당되어 운영 오버헤드가 감소한다는 점을 착안
- 무중단 Rolling 배포 방식 채택

3. Redis Keyspace Notifications 자세히

[상황 및 고려 사항]
- 해당 서비스는 경매 상태가 자동으로 경매중에서 경매 완료로 바뀌어야 하는 상황
- 실시간으로 변화한 데이터를 감지하기 위한 CDC가 필요
- 초기에는 스케줄러 사용을 고려했으나 실시간성을 보장할 수 없다는 단점이 존재
- TTL이 있는 DynamoDB, Redis, MongoDB가 고려 대상으로 선정

[결정]
- 이미 Redis를 이용한 다양한 기능(캐싱, 분산락, Redis Pub/Sub)을 사용하는 상황
- 관리할 인프라를 줄일 수 있다는 장점과 CDC, TTL이 있는 Redis 채택