

JANG
SEONG
SU

Back-end
Developer

Portfolio

장성수 (Software Engineer)

Introduce

- 개발 언어 하나만으로, 나의 아이디어를 온라인 상에서 제품화할 수 있다는 것이 개발자의 가장 큰 가치라고 생각합니다.
- 개선점이 보인다면 주저하지 않습니다. 번거로운 일을 자동화하거나, 기술을 빠르게 배워서 과감하게 도입합니다.
- 성장을 위해서는 어떤 도전 앞에서도 굴복하지 않고 끝까지 해내는 끈기와 의지를 갖추고 있습니다.
- 팀의 성장이 개인의 성장으로 귀결한다고 생각하여 함께 성장하는 것을 추구합니다.
- 현재는 ElasticSearch에 관심을 갖고 공부하고 있습니다.

Channel & Contact

- Email | wkdtdtn0219@gmail.com
- Github | <https://github.com/tjdt0219>
- Blog | <https://velog.io/@tjdt0219>

Skill

Back-end	<ul style="list-style-type: none">• Java, Spring, JPA• Python, FastAPI	Database	<ul style="list-style-type: none">• MySQL• MongoDB• ElasticSearch
DevOps	<ul style="list-style-type: none">• AWS• Github Actions, Jenkins• Docker	Tool	<ul style="list-style-type: none">• Jira• Git• Notion

Activities

외식인 기간 : 2025.06 ~ 2525.09	활동 내역 <ul style="list-style-type: none">• 36,000여 개의 프랜차이즈 가맹점이 사용하고 있는 FC다움 2.0 백엔드 개발• ElasticSearch 검색 쿼리 최적화• 레거시 데이터베이스 마이그레이션
오에스원(스타트업) 인턴 기간 : 2024.12 ~ 2025.01	활동 내역 <ul style="list-style-type: none">• 딜로이트 안진회계 법인 SI 프로젝트(회계장부관리 플랫폼) 백엔드 개발• Pandas를 활용한 데이터 병렬 처리• Openpyxl을 활용해 프로그래밍 기반 엑셀화 구현
SW마에스트로 기간 : 2024.04 ~ 2024.11	활동 내역 <ul style="list-style-type: none">• 'AI 기반 멘탈 케어 앱' 기획 개발 운영• Anroid, Back-end 개발• AI 구축 및 활용
야놀자 테크스쿨 백엔드 과정 기간 : 2023.07 ~ 2024.01	활동 내역 <ul style="list-style-type: none">• Java, Spring Framework 교육 과정 이수• 다양한 서비스의 요구사항에 따른 설계 및 개발• 야놀자 주관 기업 연계 프로젝트 수행
엘에스웨어 인턴십 기간 : 2022.07 ~ 2022.08	활동 내역 <ul style="list-style-type: none">• 사내 직원 업무를 자동화하기 위한 크롤러 자동화 시스템 개발• 크롤링 코드의 유지 보수를 위한 데이터 Guessing 전략 적용

Projects



Antifragile: AI 기반 멘탈 케어 다이어리 앱

인원 :3 기간 :2024.04 ~ 2024.11 | [Antifragile-Backend](#) | [Antifragile-Android](#) | [Play Store Download](#)

설명

Antifragile은 AI 기반 멘탈 케어 앱으로, 10월에 출시하였습니다. 사용자의 감정 일기를 AI로 분석하여 정신 건강에 도움이 되는 맞춤형 콘텐츠를 제공해주는 서비스입니다. SW마에스트로 과정에서 기획부터 개발, 출시까지 3명의 인원으로 진행하였습니다.

주 기술 스택

Kotlin / Android | Java / Spring Boot / MongoDB / MongoDB Atlas | Python / AWS / Github Actions / Docker / Pytorch

주요 역할

RAG 기반 추천 시스템 구축 [Blog | AWS Bedrock으로 RAG 구축](#)

- 헬스 케어 도메인에 특화된 AI의 정확성을 높이기 위해, 근거 기반의 응답을 제공하는 RAG 시스템 구축
- 멘탈 헬스케어 데이터를 Vector로 Embed한 후 MongoDB Atlas에 Vector Store 구축
- AWS Bedrock을 활용해 확장성을 고려한 RAG 시스템을 신속한 배포 가능하도록 설계

2만 건 데이터 자동 수집 Batch 시스템 개발 [Batch 시스템 ECS 코드](#)

- Open AI와 Youtube Data API를 활용해 2만 건의 정신 건강 관련 유튜브 콘텐츠 데이터 저장
- AWS ECS로 데이터 검색, 전처리, 저장을 각각 컨테이너화하여 배포하고, SQS를 활용해 비동기 기반으로 실행하여 시스템 가용성 향상

클라이언트 환경 On-Device LLM 구축 [PR #67 | feature/LLM-merge](#)

- 데이터 보안성을 강화하기 위해, 서버가 아닌 클라이언트 환경에서 AI를 구동하는 On-Device LLM 구축
- Mediapipe 오픈소스를 활용하여 On-Device LLM 구현
- Coroutines를 활용해 비동기 방식으로 구현하여 백그라운드 작업 최적화 및 메인 스레드 안정성 확보

Gemma 모델 QLoRA 기법 FineTuning 수행 [Blog | Gemma2B 모델 FineTuning 하기](#)

- FineTuning 수행하여 한국어 입력의 감정 추론 및 일기 요약의 정확성 15% 향상
- QLoRA 기법을 활용해 모델의 경량성 유지와 FineTuning 시 메모리 사용량 75% 이상 절감

MongoDB Clustering 구축 | LLM 모델 선정을 위한 PoC

- Spring에서 MongoDB 트랜잭션을 사용하기 위해 MongoDB Atlas 내 Replicat-Set 구축
- SK에서 제공하는 벤치마크 데이터셋 KoBest를 활용하여 LLM 성능 검증



Yanabada: 숙박 업소 중고거래 플랫폼

인원 :4 기간 :2023.12 ~ 2024.01 | [Yanabada-Backend](#) | [시연 영상](#)

설명

무료 예약 취소가 불가한 숙소의 양도/거래를 할 수 있게 해주는 플랫폼입니다. 저는 백엔드 기능 구현을 담당했습니다.

주 기술 스택

Java / Spring Boot | MySQL | AWS / Github Actions

주요 역할

예약 시스템 동시성 문제 해결 [Blog | 재고 시스템 동시성 문제 해결](#)

- 여러 사용자가 동시에 예약을 하는 경우, Exclusive Lock을 활용해 재고 수량의 데이터 정합성 보장
- Pessimistic Lock 활용하여, Connection Pool을 고려한 시스템 안정성 개선

QueryDSL을 활용한 상품 검색 기능 구현 [Commit | QueryDsl 적용](#)

- 다양한 검색 조건을 지원하기 위해 QueryDSL을 활용하여 동적 쿼리를 구현
- BooleanExpression을 이용한 조건 조합으로 코드 가독성을 개선하고, 검색 기능의 유연성 향상

JWT 기반 인증/인가 방식 구현 [Blog | 소셜로그인에 팩토리패턴 적용](#) [PR #21 | feature/jwt](#)

- Spring Security, JWT 활용해 RTR 방식의 인증 방식을 구현하여 보안성 AT, RT 탈취 시에도 보안성 강화
- 네이버, 카카오, 구글 소셜 로그인 구현 시 Factory Pattern을 활용해 확장가능한 코드 구조로 개선

Certification

- SQLD(개발자) | 2024.04
- Opic(영어) IM3 | 2025.09

Education

- 아주대학교 졸업 | 2016 - 2024

[SW Maestro] AI 기반 멘탈 케어 앱



프로젝트 소개

사용자의 감정 일기를 AI로 분석하여, 정신 건강에 도움이 되는 맞춤형 콘텐츠를 추천하는 멘탈 케어 서비스입니다.

인원 :3 기간 :2024.04 ~ 2024.11 | [Antifragile-Backend](#) | [Antifragile-Android](#) | [Play Store Download](#)

기술 스택

- Client: Android, Kotlin
- Server: Spring Boot3, Java17
- Database: MongoDB, MongoDB Atlas
- DevOps: AWS, Github Actions, Docker
- Cooperation: Git, Github, Jira, Notion

기술적 역할

- RAG 기반 추천 시스템 구축
- 2만 건 데이터 자동 수집 Batch 시스템 개발
- 클라이언트 환경 On-Device LLM 구축
- Gemma 모델 QLoRA 기법 파인튜닝 수행
- MongoDB Clustering 구축
- LLM 모델 선정을 위한 PoC
- 요구사항에 필요한 Spring 기반 Back-end API 설계 및 개발

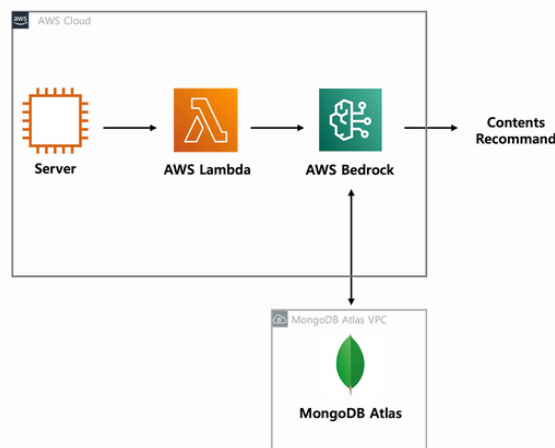
[RAG 기반 추천 시스템 구축] [Blog | AWS Bedrock으로 RAG 구축](#)

문제 상황 1

- 오픈소스 생성형 AI 만으로는 헬스케어 도메인에 특화된 정확한 응답을 제공하기 어려움
- 헬스케어 데이터의 신뢰성과 개인화된 추천 강화를 위해, 근거 기반의 응답을 제공하는 RAG기반 추천 시스템 필요

해결 1

- 헬스케어 데이터(유튜브 콘텐츠)를 Vector로 변환하고 MongoDB Atlas의 Vector Store에 저장하여, 근거 기반의 응답을 제공할 수 있도록 구축
- AWS Bedrock을 활용해, 확장성을 고려한 RAG 시스템을 신속한 배포가 가능하도록 설계



[AWS Bedrock API 아키텍처]

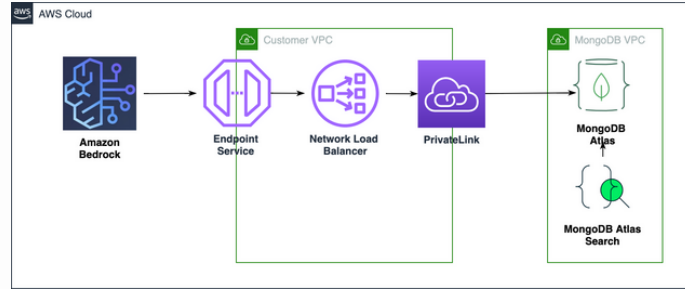
문제 상황 2

- MongoDB Cluster(서울 리전)와 AWS Bedrock(버지니아 리전)간 연동 실패

해결 2

- MongoDB Cluster를 Global Cluster로 확장하여 전 세계에서의 데이터 접근성 향상
- AWS Bedrock에 Private Link를 구축해 MongoDB Cluster에 안전하게 접근 및 보안성 강화

[SW Maestro] AI 기반 멘탈 케어 앱



[MongoDB Atlas와 VPC간 Private Link 연결]

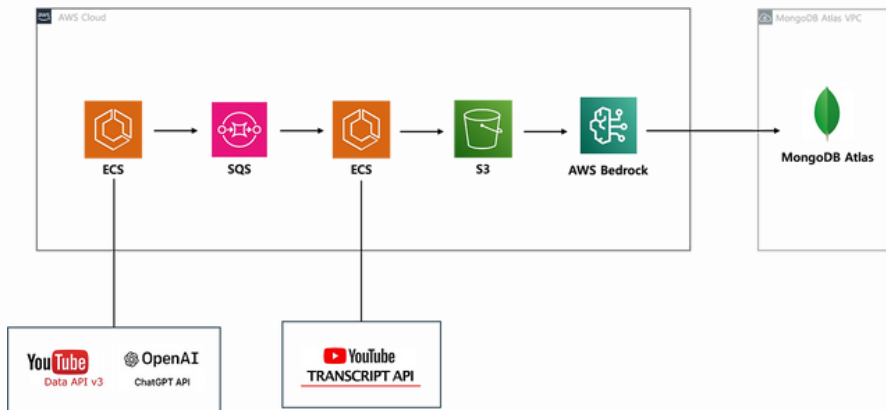
[2만 건 데이터 자동 수집 Batch 시스템 개발] [Batch 시스템 ECS 코드](#)

문제 상황

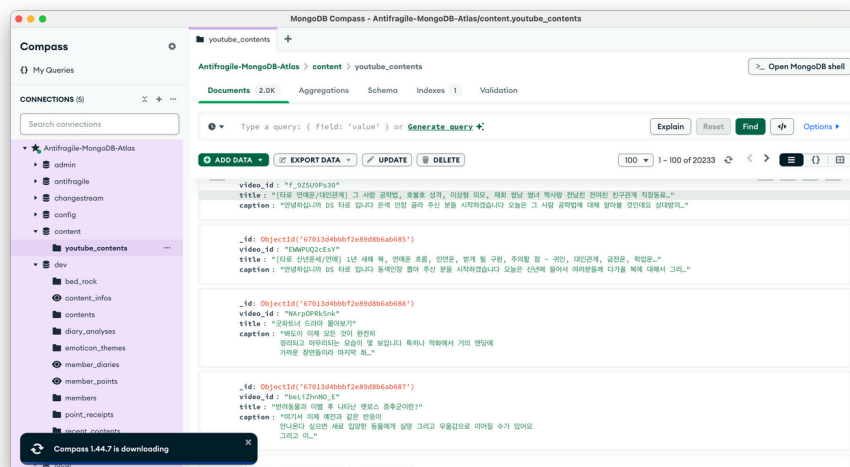
- 멘탈 케어와 관련된 대량의 콘텐츠 데이터를 직접 수집하는 데는 시간과 인력의 한계가 있음
- 수집된 데이터의 일관성과 최신성을 유지하기 어려움, 실시간으로 변화하는 멘탈 헬스케어 트렌드를 반영하기 어려움

해결

- [OpenAI와 Youtube Data API를 활용](#)하여 대규모의 헬스케어 콘텐츠 데이터를 자동으로 수집하는 Batch 시스템을 개발하여 7일만에 2만 건 데이터 수집
- [AWS ECS를 활용](#)해 데이터 검색, 전처리, 저장을 각각 컨테이너화하여 배포하고, [SQS를 활용](#)해 비동기 처리로 시스템 가용성을 향상시켜 데이터 최신성 유지



[콘텐츠 데이터 수집 배치 시스템 아키텍처]



[2만개 콘텐츠 수집 캡처본]

[SW Maestro] AI 기반 멘탈 케어 앱



[On-Device LLM 구축] [PR #67 | feature/LLM-merge](#)

문제 상황

- 감정 일기와 같은 민감한 데이터를 서버에 저장하면 보안 문제가 발생할 수 있음
- 서버 기반으로 시를 운영할 경우, 지속적인 인프라 비용이 발생하여 비용 부담이 크고, 사용량 증가에 따라 서버 확장이 필요

해결

- 서버가 아닌 클라이언트 환경에서 시를 구동하는 On-Device LLM을 구축하여, 사용자 데이터를 로컬에서 처리하도록 설계
- 클라이언트 디바이스에서 직접 시를 실행함으로써 서버 운영 비용을 절감하고, 추가적인 서버 확장 없이도 효율적인 AI 서비스 제공이 가능하도록 최적화

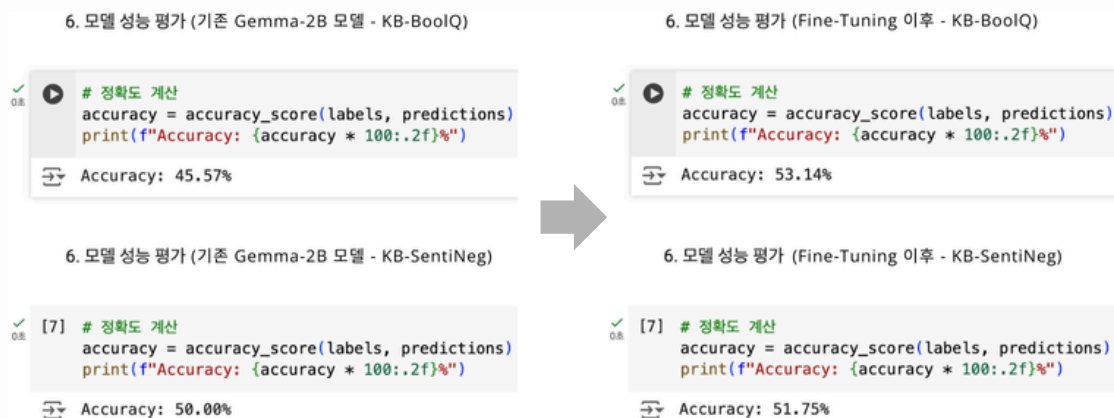
[Gemma 모델 QLoRA 기법 FineTuning 수행] [Blog | Gemma2B 모델 FineTuning 하기](#)

문제 상황

- 기존 Gemma 모델은 한국어 입력에 대한 감정 추론 및 일기 요약의 정확도가 낮아, 실제 사용자에게 신뢰도 높은 결과를 제공하기 어려움
- 일반적인 Fine-Tuning 과정에서는 고용량 GPU 메모리가 필요하여, 비용과 자원 효율성 측면에서 부담이 큼

해결

- QLoRA 기법을 활용해 Gemma 모델을 Fine-Tuning하여, 한국어 입력의 감정 추론 및 일기 요약 정확도를 15% 향상
- QLoRA를 적용함으로써 모델의 경량성을 유지하면서도, Fine-Tuning 시 메모리 사용량을 75% 이상 절감하여 효율적인 학습 환경 구축

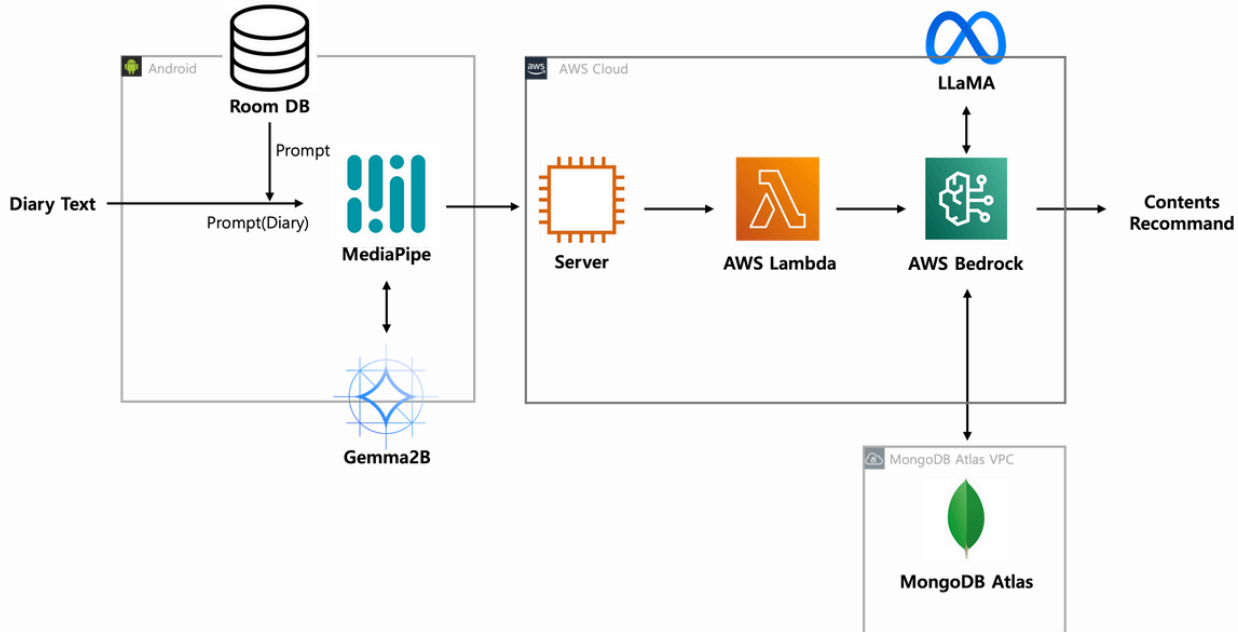


[파인 튜닝 전/후 성능 측정 비교]

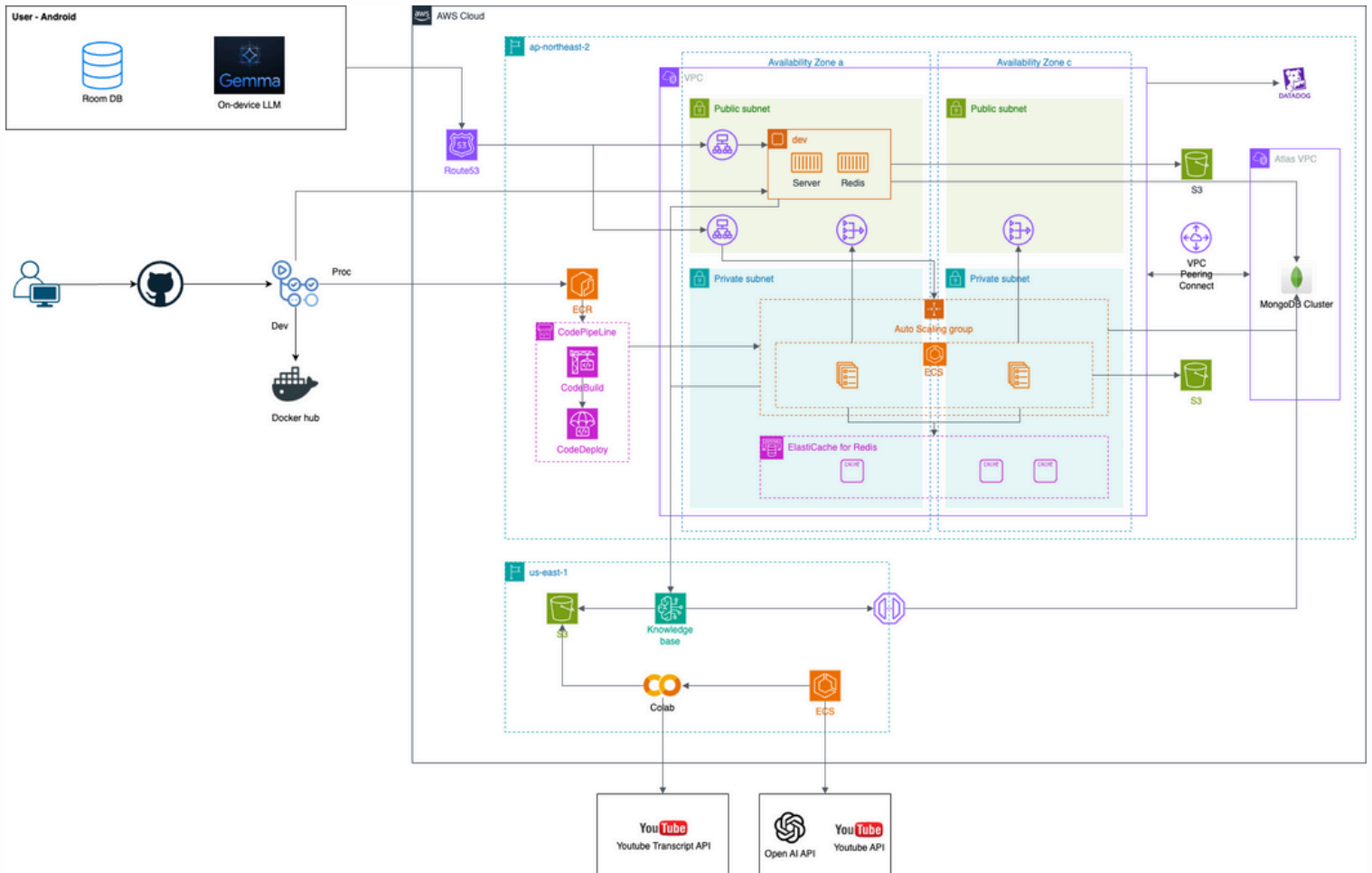
[SW Maestro] AI 기반 멘탈 케어 앱



시스템 플로우 (On-Device LLM + RAG 추천 시스템)



인프라 아키텍처



프로젝트 소개

무료 예약 취소가 불가한 숙박업소를 양도 및 중고거래하는 서비스입니다.

인원 :4 기간 :2023.12 ~ 2024.01 | [Yanabada-Backend](#) | [시연 영상](#)

기술 스택

- Server: Spring Boot3, Java17
- Database: MySQL
- DevOps: AWS, Github Actions, Docker
- Cooperation: Git, Github, Notion

기술적 역할

- Git Flow 전략 도입
- 예약 시스템 동시성 문제 해결
- QueryDSL을 활용한 상품 검색 기능 구현
- Spring Security + JWT를 활용한 인증 인가 구현
- Factory Pattern을 활용한 소셜 로그인 구현
- JUnit5을 활용해 테스트코드 구현

[예약 시스템 동시성 문제 해결]

[Blog | 재고 시스템 동시성 문제 해결](#)

문제 상황

- 여러 사용자가 동시에 예약을 시도할 경우, 재고 수량이 정확하게 반영되지 않아 중복 예약이나 초과 예약이 발생할 위험이 있음
- 높은 동시 요청이 발생할 때, 데이터 정합성을 유지하는 과정에서 시스템의 성능 저하 또는 데드락과 같은 안정성 문제가 발생할 수 있음

해결

- Exclusive Lock을 활용하여 동시 예약 시에도 재고 수량의 데이터 정합성을 보장하고, 중복 예약 및 초과 예약 문제를 방지
- Pessimistic Lock을 적용하여 Connection Pool을 고려한 효율적인 동시성 제어를 수행하고, 시스템 안정성을 향상

문제 해결 시도 1: Java의 Synchronized

```
@Transactional 2 usages
public synchronized void decreaseSync(Long id, Long quantity) {
    Stock stock = stockRepository.findById(id).orElseThrow();
    stock.decrease(quantity);

    stockRepository.saveAndFlush(stock);
}
```



Expected :0L
Actual :49L

[동시성 테스트 실패]

synchronized를 명시하면, decrease 메소드에 하나의 스레드만 접근이 가능하므로, 데이터의 동시성 문제를 해결할 것이라고 기대
그러나 @Transactional의 AOP 특성으로 인해, **트랜잭션 종료 시점과 synchronized 메소드의 종료 시점의 불일치** → 테스트 실패

문제 해결 시도 2: Pessimistic Lock

```
@Lock(LockModeType.PESSIMISTIC_WRITE) 1 usage
@Query("select s from Stock s where s.id = :id")
Stock findByIdWithPessimisticLock(Long id);
```



Test Results 1 sec 488 ms
StockServiceTest 1 sec 488 ms
동시에 100개의 요청으로 재고를 감소시킨다. 1 sec 488 ms

[동시성 테스트 성공]

예약 시스템은 데이터 정합성의 **충돌이 많이 발생할 것으로 예상**하여, 낙관적 락 대신에 **비관적 락을 사용하기로 결정**
JPA를 통해 **X-Lock을 비교적 쉽게 구현이 가능**하고, 데이터에 직접 Lock을 거는 것이기 때문에 **동시성 문제를 완전히 해결**

그러나 Pessimistic Lock은 DB 자체에 Lock을 거는 쿼리가 발생하기 때문에, **DB Connection 비용과 Disk I/O를 발생하여 성능이 감소**하고,
다른 트랜잭션의 해당 데이터에 대한 **Read/Write에 대해 블로킹이 발생하여 성능 저하 발생**

문제 해결 시도 3: Redis Redisson의 분산락

```
@Component 1 usage
@RequiredArgsConstructor
@AllArgsConstructor
public class RedissonLockStockFacade {

    private RedissonClient redissonClient;

    private StockService stockService;

    public void decrease(Long id, Long quantity) {
        RLock lock = redissonClient.getLock(id.toString());
        try {
            boolean available = lock.tryLock(10, 10, TimeUnit.SECONDS);
            if(!available) {
                System.out.println("lock 획득 실패");
                return ;
            }
            stockService.decrease(id, quantity);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        } finally {
            lock.unlock();
        }
    }
}
```



975 ms	Expected :101L
975 ms	Actual :65L

[동시성 테스트 실패]

Redis의 Redisson은 pub/sub 구조로 Lock을 구현하기 때문에, 성능 측면에서 다른 Lock보다 뛰어남
그러나, **트랜잭션 종료 시점과 Lock의 해제 시점이 불일치하면 데이터의 정합성이 무너지는** 문제가 발생함

이를 해결하기 위해, **Propagation.REQUIRES_NEW 옵션을 사용해 트랜잭션을 분리** → 데이터 정합성 보장

그러나 이 방법도 아래와 같은 문제가 있을 것이라고 판단

- REQUIRES_NEW을 사용하면 하나의 작업에서 Connection을 2개를 사용 → **Connection Pool 사이즈가 작다면 데드락 발생**
- 트래픽을 예상할 수 없었기에, **적절한 Connection Pool 사이즈를 설정할 수 없는 상황** → 서비스 전체 장애 발생

결론

Redisson의 Side-Effect를 해결하면서까지 적용할만한 당장의 성능 문제가 아니라고 생각했기 때문에, 구현이 간단한 **비관적 락으로 결정**

[QueryDSL을 활용한 상품 검색 기능 구현]

[Commit | QueryDsl 적용](#)

문제 상황

- 기존 SQL 기반의 상품 검색 기능은 고정된 조건만을 지원하여, 다양한 검색 조건을 적용하는 데 한계가 있음
- 복잡한 검색 로직이 증가할수록 코드 가독성이 저하되고 유지보수가 어려워지는 문제가 발생

해결

- QueryDSL을 활용하여 동적 쿼리를 구현함으로써, 사용자 요구에 맞는 다양한 검색 조건을 지원하도록 개선
- BooleanExpression을 이용해 조건을 조합하여 코드의 가독성을 높이고, 유지보수가 용이한 구조로 검색 기능을 확장

[JWT 인증/인가 방식 구현 | 소셜 로그인 구현]

[Blog | 소셜로그인에 팩토리패턴 적용](#)

[PR #21 | feature/jwt](#)

문제 상황

- 기존 인증 방식에서는 액세스 토큰 또는 리프레시 토큰이 탈취될 경우 보안 위협이 발생할 가능성이 있음
- 네이버, 카카오, 구글 등의 소셜 로그인 기능을 추가할 때마다 인증 로직이 분산되어 코드 유지보수가 어려움

해결

- Spring Security와 JWT 기반의 RTR(Rotate Refresh Token) 방식 인증을 적용하여, RT가 탈취되더라도 재사용을 방지하고 보안성을 강화
- Factory Pattern을 적용해 소셜 로그인 인증 로직을 확장 가능하게 구조화하여, 새로운 인증 공급자 추가 시 코드 수정 없이 유연하게 대응 가능

시스템 아키텍처

