PORTFOLIO

개발의 가치를 전달하는 개발자

Phone

010-8079-5427

Email

kkl5468@gmail.com

Blog

https://kimgyeonglock.github.io

INTRODUCTION

안녕하세요, 김경록입니다.

EXPERIENCE

2019.02 한동대학교 입학
AI·컴퓨터공학심화 전공

2022.12 **<한밥>** , **<반디>** 앱 개발 및 출시
2023.08 플레이스토어, 앱스토어 출시 완료

2023.06 **Nein to Sick** 창업 CTO

2024.12 **카카오테크 부트캠프** 풀스택 1기 수료

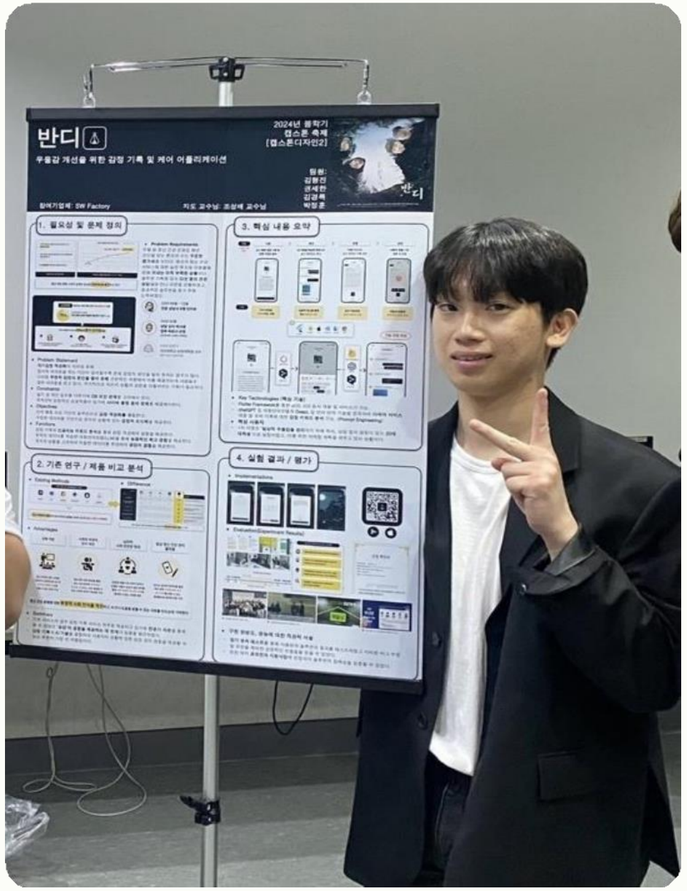
2025.08 **삼성전자 DX 부문 S/W 알고리즘 역량 강화 특강** 수료

4.17/4.5
전체 학점

280명
누적 다운로드 수

6,000만원
지원금 확보

SAMSUNG 삼성전자



AWARDS

수상 내역

총 10개 수상 경력 보유 (프로그래밍, 프로젝트)


기독일보

한동대, 제13회 RPM 창업경진대회 성료

목회·신학 | 신학 장요한 기자 press@cdaily.co.kr

입력 2024. 11. 26 06:29

AI 기반 창업 아이디어로 청년 혁신가들 열정 빛나다

제13회 RPM 창업경진대회 참여 학생들 기념 사진. ©한동대 제공

대상은 AI 기반 감정 일지 플랫폼 '반디'를 개발한 Nein to Sick 팀(팀장 김서현)이 차지했다. 이 플랫폼은 사용자의 감정을 데이터로 기록하고 관리할 수 있는 서비스로, 독창성과 실용성을 인정받아 심사위원들로부터 높은 평가를 받았다.


기독일보

한동대, 제12회 한동 창업아이디어 경진대회 개최

목회·신학 | 신학 장지동 기자 zidgilove@cdaily.co.kr

입력 2023. 12. 05 12:17

'Pheonix Labs' 팀 1위 선정

제12회 한동창업아이디어경진대회 참가팀 단체 사진. ©한동대 제공

본선에 진출한 참가팀은 △대형 언어모델(LLM) 기반의 일상 감정 관리 플랫폼인 'Nein To Sick 팀' △공연예술 구독 서비스 플랫폼을 제안한 '알터즈 팀' △700만 최대 디아스포라를 대상으로 한 원스톱 라이프스타일 플랫폼


세상을 보는 키워드 미디어 경북

홈 사회 생활·문화 교육 정치·경제 연예·스

홈 > 교육

한동대 SW중심대학 4개 창업팀, 2024 K-startup 지원사업 최종 선발

한동대학교 SW중심대학 4개 창업팀과 조성배 지도교수

한동대 학부생으로 구성된 △ NTS팀(김형진,김세한,김경록,박창휘,서지은,이지원) 데이터 기반 일상 감정 관리 플랫폼 '반디' △E2I팀(정수아,박승리,조

제 13회 창업경진대회 RPM 대상

2024-12

캡스톤디자인 경진대회 우수상

2024-06

대경권 프로그래밍 경진대회 우수상

2024-05

POSTECH Mini-I-Corps 우수상

2024-02

제 12회 창업경진대회 RPM 장려상

2023-11

SW Festival 스마트어플리케이션 부문 대상

2023-11

SW Festival 문제해결 아이디어 공모전 장려상

2023-11

SW 창업 경진대회 대상

2023-10

ACM-ICPC 경진대회 동상

2023-10

SW Festival C Programming 콘테스트 우수상

2019-12

PERSONAL SKILLS

핵심 역량

CERTIFICATIONS

- TOEIC Speaking IH
  2025-08
- 리눅스 마스터 2급
  2022-04
- 네트워크 마스터 2급
  2022-04

CHANNELS

- Github - https://github.com/KimGyeongLock
- Blog - https://kimgyeonglock.github.io/

PROJECTS

LLM 기반 일상 감정 관리 앱, 반디

비대면 중고거래 웹 서비스, 거래함

실시간 텍스트 통화 웹 서비스, 앵무말

배달 음식 공동 구매 앱, 한밥

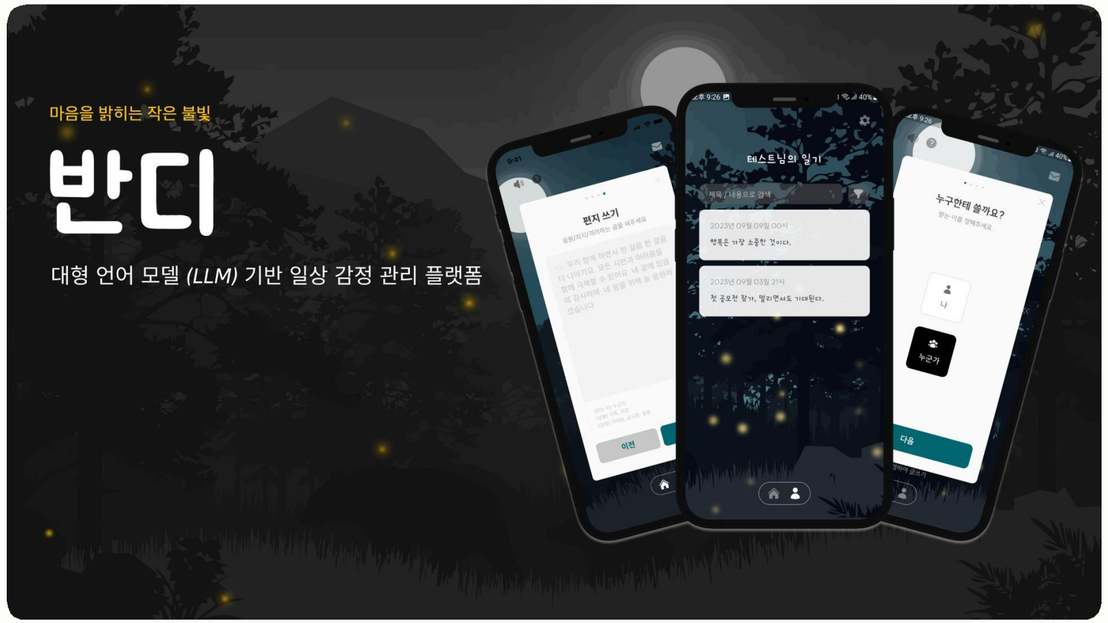
MOBILE APP PROJECT

반디 (백엔드 & 프론트엔드 개발)

2023-08 ~ **운영중 (280명의 누적사용자 달성)**

주제

'반디'는 우울과 같은 정신 건강 문제를 겪는 유저들을
대상으로 일상적 감정 관리를 도울 수 있는 어플리케이션으로,
LLM ChatGPT API를 사용하여 감정 분석 기능을 제공하며
도출된 키워드를 통해 유저 간의 커뮤니케이션을 조성합니다.



개인 기여 (CTO) – 6인 팀

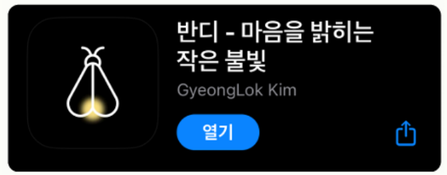
- 일기 작성 및 공유 기능 중심 개발
- 대표 문서 체계 도입해 데이터 탐색 효율 개선
- Sentry 기반 에러 수집 및 모니터링 시스템 구축
- B2B 확장 후 고객사 전용 통계 대시보드, 민원 게시판 개발
- DeepL API 기반 번역 기능 및 다국어화 적용

Flutter 3.19.3 Notion

OpenAI API
(ChatGPT 4.0 Turbo)

Github

Firebase



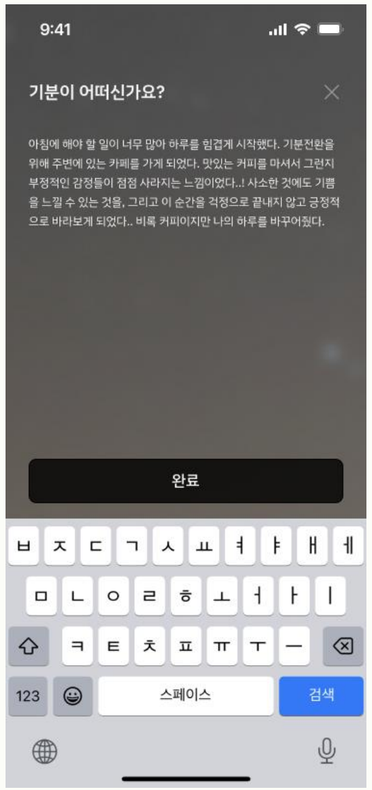
https://tosto.re/bandiApp

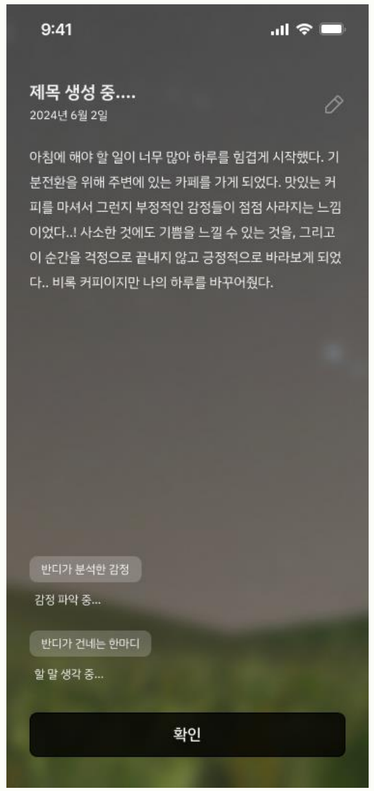
https://github.com/Nein-to-Sick/bandi_official

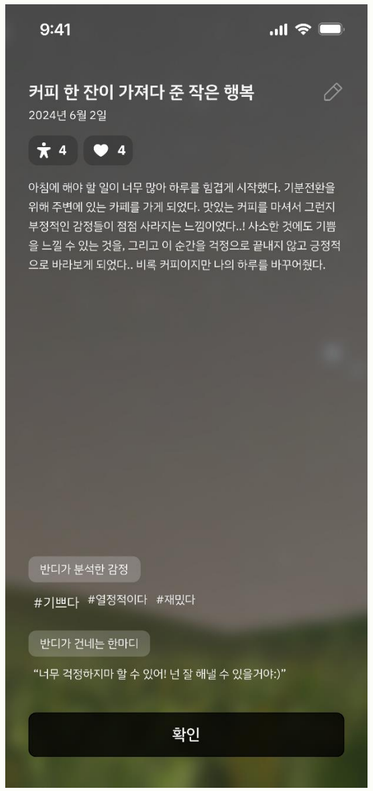
DETAILS

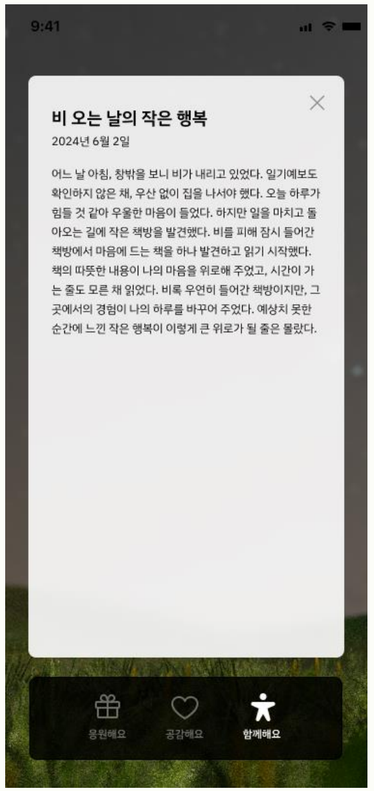
상세 설명

일기 작성 및 공유 로직 구현









사용자는 일기를 작성한 후, **AI 기반 감정 분석과 응원의 메시지**를 받을 수 있습니다.
감정 키워드는 총 **5개의 카테고리로 구성되어 있으며, 90가지 세부 감정 키워드**를 보유하고 있습니다.
일기 작성이 완료되면, 사용자는 **다른 사용자의 일기를 열람할 수 있으며,**
이때 작성 시간과 감정 상태를 고려해 **사용자 맞춤형 일기 콘텐츠**가 제공됩니다.

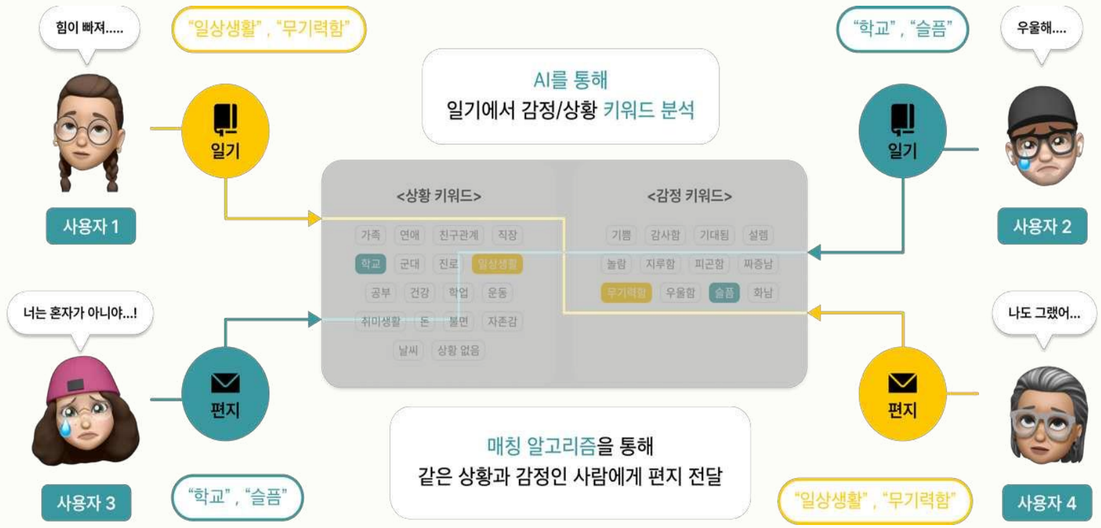
DETAILS

상세 설명

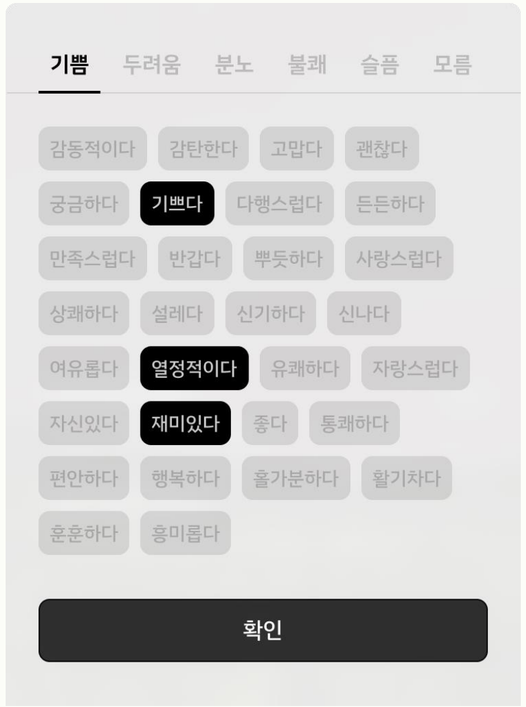
감정별 대표 문서 체계 도입

문제 정의

기존 유저 커뮤니케이션 알고리즘의 경우,
상황과 감정 키워드의 일치율의 계산하여 사용자에게 맞는 일기를 전달



AI 정확도를 높이기 위하여
감정 키워드 **90개**로 확장



5개의 카테고리화 적용하여
일기의 감정 카테고리의 일치로 변경

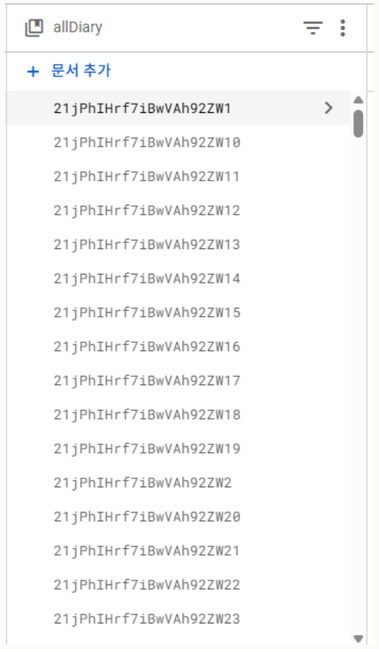
DETAILS

상세 설명

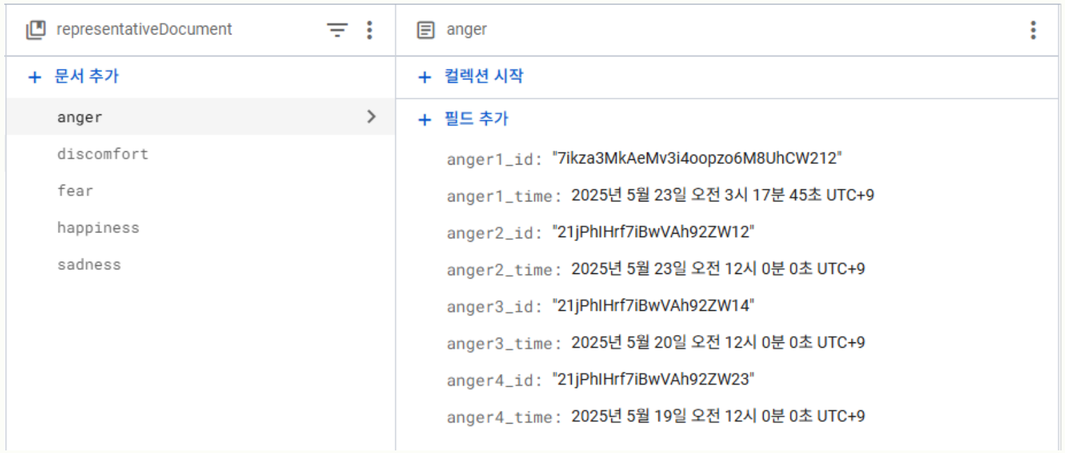
감정별 대표 문서 체계 도입

문제 정의

사용자 수가 증가함에 따라 저장된 일기 데이터가 500개를 초과하게 되었고, 맞춤형 일기를 선택하기 위해 **모든 데이터를 조회하는 비효율적인 Read 연산**이 발생



문제 해결



감정 카테고리별로 **4개의 대표 문서**를 생성
추출된 키워드와 유효시간을 계산하여 **가장 유사한 감정 카테고리** 탐색
일주일을 기준으로 대표 감정 일기를 **최신화**

결과: O(N) → O(5 × 4) 데이터 탐색 최적화

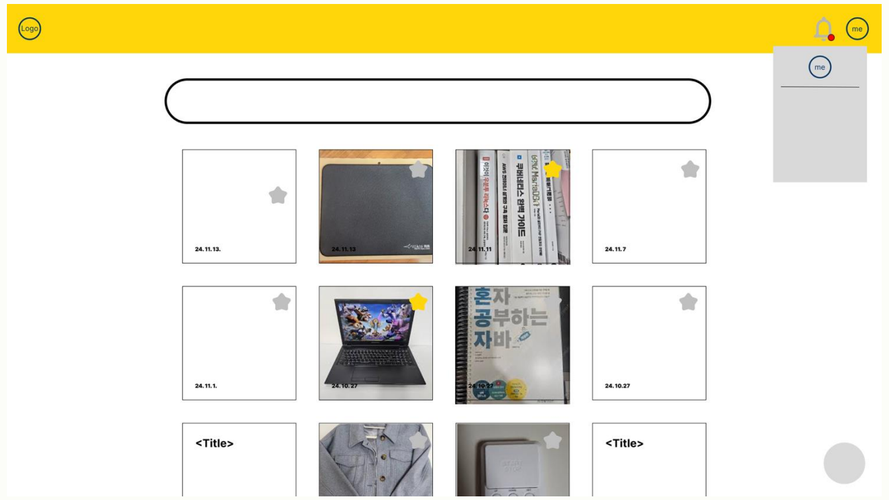
WEB PROJECT

거래함 (백엔드 개발)

2024-10 ~ 2024-12

주제

'거래함'은 카카오테크 부트캠프 내에서 사물함을 이용한 중고거래 플랫폼입니다. 사용자들은 판매할 물건을 사물함에 보관하고, 구매자는 해당 사물함에서 물건을 찾아가는 방식으로 거래가 이루어집니다.



프론트 데모 페이지

개인 기여 (Backend Developer)

페어 프로그래밍으로 진행

- Spring Oauth2 Client 기반 로그인 구현
- 판매/구매/조회 기능, 알림 및 검색 기능 구현
- 좋아요, 조회수 데이터를 Redis에 캐싱, 주기적 DB 동기화
- 구매 로직에 Pessimistic Lock 적용하여 동시성 문제 해결
- N + 1 문제 Fetch Join을 활용해 해결

Spring Boot 3.3.1

MySQL 8.3.0

redis

https://github.com/Trade-Ham

DETAILS

상세 설명

Race Condition 문제 해결

문제 정의

Unit Test를 진행하여 멀티 스레드 환경에서 구매 로직을 진행
임의로 100개의 요청을 32개의 쓰레드에서 실행
총 87개의 요청만 정상적으로 출력하고 23개의 요청이 작동하지 않음을 확인

```
expected: <0> but was: <87>
Expected :0
Actual   :87
```

Pessimistic Lock vs Optimistic Lock

- Optimistic Lock 적용 시 충돌 발생 시점마다 재시도 로직이 반복
- 구매 로직처럼 충돌이 잦은 환경에서는 성능 저하 발생 가능성 확인
- **Pessimistic Lock**을 DB 레벨에 적용해 동시성 이슈 예방

```
2024-11-02T11:53:45.735+09:00 DEBUG 23891 --- [ool-2-thread-10] org.hibernate.SQL
Hibernate: select s1_0.id,s1_0.product_id,s1_0.quantity from stock s1_0 where s1_0.id=? for update
```

Redis

추후 로그인 단계와 데이터 탐색 최적화를 위해 Redis를 도입
Redis의 **분산 Lock**으로 원자성 보장 기능을 활용해 동시성 문제를 동시에 해결

DETAILS

상세 설명

좋아요/조회수 소셜 기능 성능 및 동시성 최적화

문제 정의

- 좋아요, 조회수와 같은 소셜 기능은 사용자 상호작용이 많아 **DB Write 부하 및 동시성 문제가 빈번히 발생**
- 사용자 수 증가에 따라 성능 저하 및 데이터 무결성 문제 발생 가능성 확인

1. 찜하기 기능 최적화

- **좋아요 중복 요청 처리**
    - 애플리케이션 레벨에서 existingLike 검사를 통해 중복 요청 논리적 차단
    - 데이터베이스 레벨에서 user_id + product_id 조합에 Unique 제약 조건 추가로 중복 방지
- **성능 문제**
    - Redis를 도입해 좋아요 정보를 인메모리 저장소에서 처리하여 DB 접근 최소화
    - Redis에 저장된 좋아요 수를 주기적으로 DB에 동기화 (임의 1분 주기, 유동적 조정 가능)
- **트랜잭션 경쟁 상태 (Race Condition)**
    - Redis의 원자적 연산(INCR/DECR)을 활용해 동시 요청에 대한 데이터 정합성 보장
- **삭제된 상품 처리**
    - 상품 삭제 전 관련 좋아요 데이터 선삭제 처리 (likeRepository.deleteByProduct 사용)

DETAILS

상세 설명

좋아요/조회수 소셜 기능 성능 및 동시성 최적화

문제 정의

- 좋아요, 조회수와 같은 소셜 기능은 사용자 상호작용이 많아 **DB Write 부하 및 동시성 문제가 빈번히 발생**
- 사용자 수 증가에 따라 성능 저하 및 데이터 무결성 문제 발생 가능성 확인

2. 조회수 기능 최적화

- **유저당 1회 조회 제한**
  - 유저가 동일 상품을 반복 조회해도 조회수가 중복 증가하지 않도록 제한 필요
  - Redis의 **Set 자료구조**를 활용하여 "product:{productId}:viewed_users" 형태로 유저 ID 저장
  - SADD 명령어를 활용해 처음 조회한 유저만 조회수 증가
  - Set 데이터는 **7일** 후 만료되도록 설정하여 이후 재방문 시 조회수 재반영 가능
- **성능 최적화**
  - Redis를 조회수 캐시 저장소로 활용, 빠른 조회와 증가 처리 구현
  - 일정 주기로 Redis에서 DB로 조회수를 반영하는 배치 프로세스 구성
  - Redis의 INCR 명령은 단일 스레드 기반으로 원자성을 보장하므로 동시성에도 안전함

DETAILS

상세 설명

N + 1문제 개선

문제 정의

- 매물로 등록된 **상품 데이터**를 모두 조회하고자 할 때, 상품에 연관된 **사용자**, **거래 내역** 등의 테이블과의 **연관관계 조회**가 함께 발생
- 상품 수가 많아질수록 동일한 쿼리가 반복 실행되며, 이는 **N+1 문제**로 이어짐
- 결과적으로 **상품 수가 증가할수록 데이터베이스 부하가 선형적으로 증가**하고, 페이지 응답 속도 및 전체 서비스 성능에 악영향을 미침

```
2024-12-14T19:53:46.394+09:00 DEBUG 20442 --- [Trade-Ham] [           main] org.hibernate.SQL                        : select pe1_0.productId,pe1_0.buyer_id,
 pe1_0.createdAt,pe1_0.description,pe1_0.likes,pe1_0.locker_id,pe1_0.modifiedAt,pe1_0.name,pe1_0.price,pe1_0.seller_id,pe1_0.status,pe1_0.views from ProductEntity
  pe1_0 where pe1_0.status=? order by pe1_0.createdAt desc
2024-12-14T19:53:46.410+09:00 DEBUG 20442 --- [Trade-Ham] [           main] org.hibernate.SQL                        : select ue1_0.id,ue1_0.acount,ue1_0.email,
 ue1_0.nickname,ue1_0.profile_image,ue1_0.provider,ue1_0.realname,ue1_0.role,ue1_0.username from UserEntity ue1_0 where ue1_0.id=?
2024-12-14T19:53:46.425+09:00 DEBUG 20442 --- [Trade-Ham] [           main] org.hibernate.SQL                        : select ue1_0.id,ue1_0.acount,ue1_0.email,
 ue1_0.nickname,ue1_0.profile_image,ue1_0.provider,ue1_0.realname,ue1_0.role,ue1_0.username from UserEntity ue1_0 where ue1_0.id=?
2024-12-14T19:53:46.427+09:00 DEBUG 20442 --- [Trade-Ham] [           main] org.hibernate.SQL                        : select ue1_0.id,ue1_0.acount,ue1_0.email,
 ue1_0.nickname,ue1_0.profile_image,ue1_0.provider,ue1_0.realname,ue1_0.role,ue1_0.username from UserEntity ue1_0 where ue1_0.id=?
2024-12-14T19:53:46.428+09:00 DEBUG 20442 --- [Trade-Ham] [           main] org.hibernate.SQL                        : select ue1_0.id,ue1_0.acount,ue1_0.email,
 ue1_0.nickname,ue1_0.profile_image,ue1_0.provider,ue1_0.realname,ue1_0.role,ue1_0.username from UserEntity ue1_0 where ue1_0.id=?
2024-12-14T19:53:46.429+09:00 DEBUG 20442 --- [Trade-Ham] [           main] org.hibernate.SQL                        : select ue1_0.id,ue1_0.acount,ue1_0.email,
 ue1_0.nickname,ue1_0.profile_image,ue1_0.provider,ue1_0.realname,ue1_0.role,ue1_0.username from UserEntity ue1_0 where ue1_0.id=?
2024-12-14T19:53:46.430+09:00 DEBUG 20442 --- [Trade-Ham] [           main] org.hibernate.SQL                        : select ue1_0.id,ue1_0.acount,ue1_0.email,
 ue1_0.nickname,ue1_0.profile_image,ue1_0.provider,ue1_0.realname,ue1_0.role,ue1_0.username from UserEntity ue1_0 where ue1_0.id=?
2024-12-14T19:53:46.431+09:00 DEBUG 20442 --- [Trade-Ham] [           main] org.hibernate.SQL                        : select ue1_0.id,ue1_0.acount,ue1_0.email,
 ue1_0.nickname,ue1_0.profile_image,ue1_0.provider,ue1_0.realname,ue1_0.role,ue1_0.username from UserEntity ue1_0 where ue1_0.id=?
2024-12-14T19:53:46.433+09:00 DEBUG 20442 --- [Trade-Ham] [           main] org.hibernate.SQL                        : select ue1_0.id,ue1_0.acount,ue1_0.email,
 ue1_0.nickname,ue1_0.profile_image,ue1_0.provider,ue1_0.realname,ue1_0.role,ue1_0.username from UserEntity ue1_0 where ue1_0.id=?
2024-12-14T19:53:46.434+09:00 DEBUG 20442 --- [Trade-Ham] [           main] org.hibernate.SQL                        : select ue1_0.id,ue1_0.acount,ue1_0.email,
 ue1_0.nickname,ue1_0.profile_image,ue1_0.provider,ue1_0.realname,ue1_0.role,ue1_0.username from UserEntity ue1_0 where ue1_0.id=?
2024-12-14T19:53:46.435+09:00 DEBUG 20442 --- [Trade-Ham] [           main] org.hibernate.SQL                        : select ue1_0.id,ue1_0.acount,ue1_0.email,
 ue1_0.nickname,ue1_0.profile_image,ue1_0.provider,ue1_0.realname,ue1_0.role,ue1_0.username from UserEntity ue1_0 where ue1_0.id=?
Number of queries executed: 11


org.opentest4j.AssertionFailedError: N+1 문제가 발생하지 않도록 하나의 쿼리로 데이터를 가져와야 합니다. ==>
Expected :1
Actual   :11
```

1

N(10)

해결 방법

실제 테스트: 거래함 서비스의 **8**개 GET API에 대해 테스트 코드를 작성하고 실행하여 N+1 문제가 발생하는지 확인

API 명세서

API	Method	uri	N+1 문제가 있는가?	해결방법
판매 중인 전체 물건 조회	GET	/api/v1/product/all	Yes	Fetch Join
물건 디테일 1개 조회	GET	/api/v1/product/{product_id}/detail	No	
물건 구매 버튼 클릭	GET	/api/v1/product/purchase-page/{product_id}	No	
물건 검색	GET	/api/v1/product/search	No	
구매 내역 조회	GET	/api/v1/my/purchase	Yes	Fetch Join
판매 내역 조회	GET	/api/v1/my/sell	No	
사용자가 좋아요한 물품 조회	GET	/api/v1/my/likes	No	
알림 조회	GET	/api/v1/notifications	No	

+ 새 페이지

해결방안

1. Fetch join
2. EntityGraph

Fetch Join vs EntityGraph

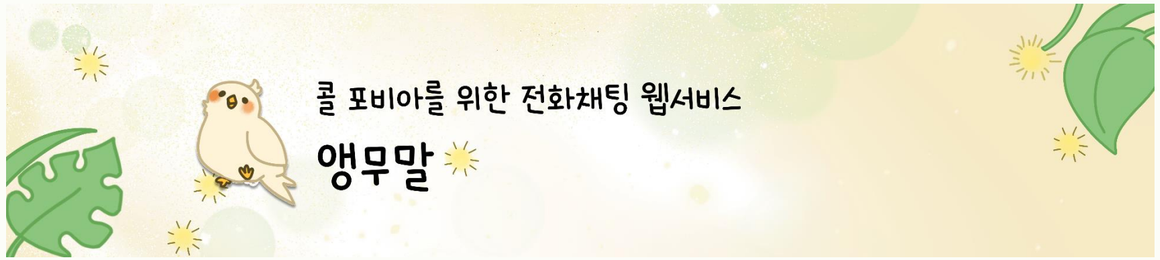
3. BatchSize
4. QueryDSL

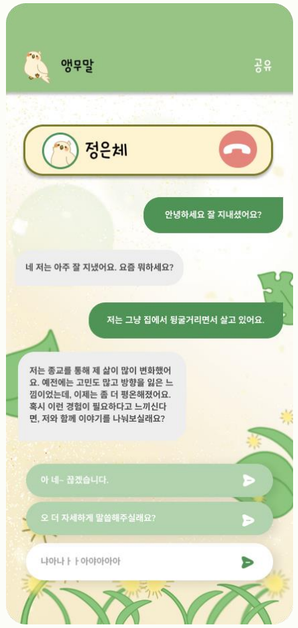
Fetch Join, **EntityGraph**, **BatchSize**, **QueryDSL**을 직접 적용하고 응답 시간을 비교하여 적절한 해결방법을 선정

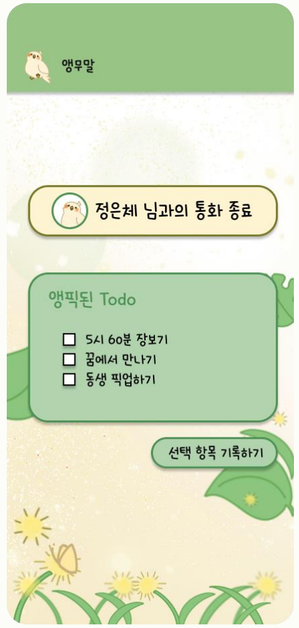
WEB PROJECT

앵무말 (백엔드 & 프론트엔드 개발)

2024-09 ~ 2024-12







주제

'앵무말'은 전화 통화에 어려움을 겪는 것을 일컫는 '콜포비아' 극복을 위한 AI 기반 통화 보조 서비스입니다.
음성 통화를 텍스트로, 텍스트를 AI 음성 기술로 변환하여 사용자와 상대방 간의 의사소통을 더욱 편리하고 효과적으로 만들어 줍니다.

개인 기여 (Full-Stack Developer) - 6인 팀

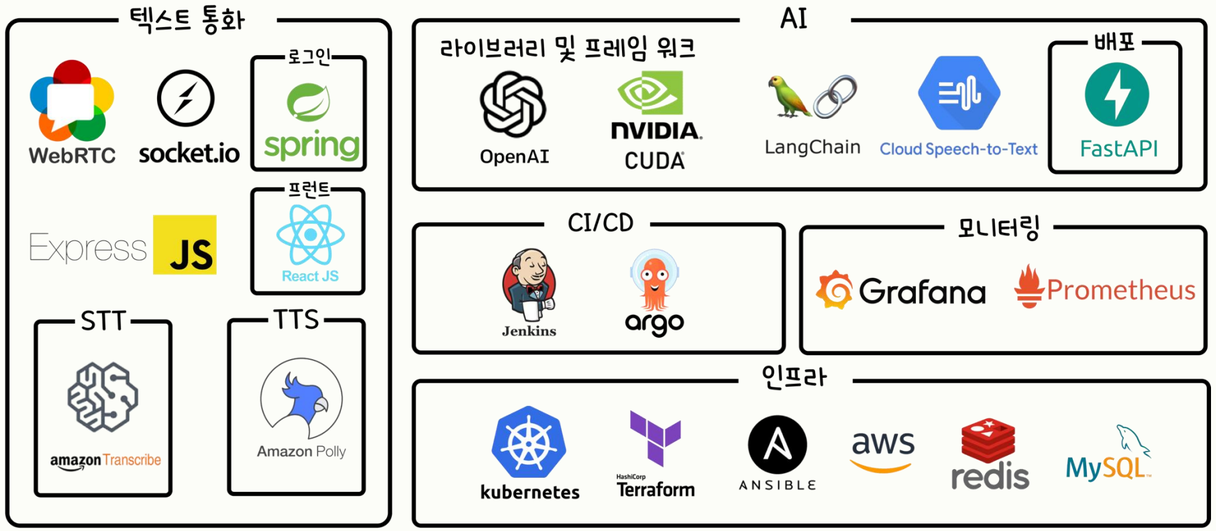
- OAuth2 로그인, JWT 인증, DB 설계, WebRTC 기반 통화 환경 구축
- Todo 기능 CRUD, Paging 적용
- AWS STT/TTS 연동해 실시간 음성 인식 및 변환 파이프라인 구현
- Playwright & Artillery기반 E2E 테스팅

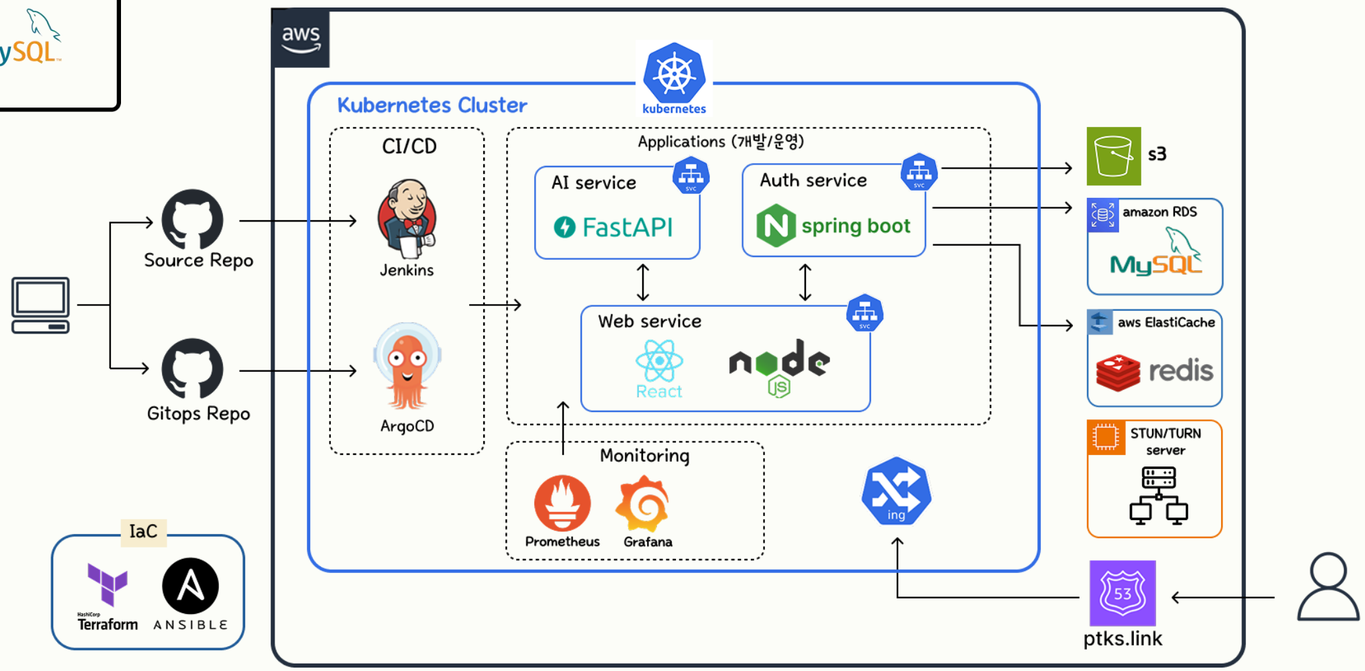
https://github.com/Parrotalk

시연영상

DETAILS

기술 스택 & 아키텍처





DETAILS

상세 설명

WebRTC 기반 STT 변환

문제 정의

- Stream 타입의 오디오 데이터를 STT로 변환하는 과정에서 참고할 자료가 부족했으며, 관련 지식도 부족한 상황
- AWS Transcribe 서비스를 사용해 음성 데이터를 인코딩하려 했으나, **text response 값이 null로 반환되는 문제**가 발생하며 인코딩 과정에서 오류가 있음을 확인

해결 방법

- AWS Transcribe 공식 문서의 Best Practices를 참고하여, WebRTC에서 요구하는 음성 데이터 형식인 **16비트 PCM** 형식으로 변환
- 브라우저 샘플 레이트를 **48kHz**로 설정하여 문제를 해결하고 정상적으로 구동
- 지연율에 따라 비례하는 audio chunk 크기를 추가적으로 조절

결과

- 입력된 문장에서 첫 모음만 나열되던 초기 결과에서, 보이스와 동일한 대화 텍스트를 정확하게 추출하며 **정확도 향상**
- 초기 지연시간이 **3~5분**에서 **20~30초**로 크게 감소

MOBILE APP PROJECT

한밥 (백엔드 & 프론트엔드 개발)

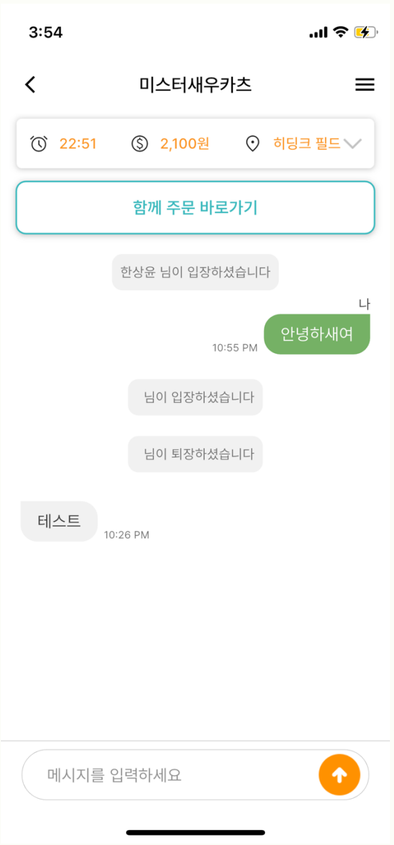
2022-12 ~ 2025-02

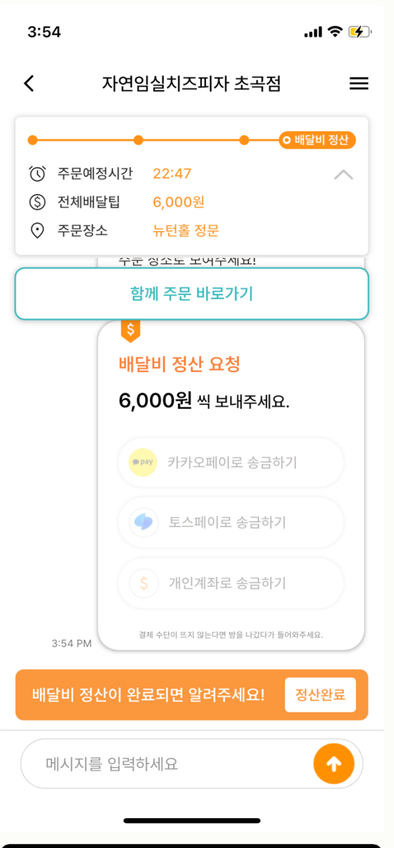
주제

'한밥'은 인상된 배달비에 부담을 느끼는 기숙사생, 1인 가구를 위한 배달 공동 구매 어플리케이션입니다. Stream을 사용한 실시간 채팅 기능과 배달의 민족, 카카오페이, 토스 어플의 URL을 연결시켜 통합된 커뮤니티를 형성하였습니다.

개인 기여 (Team Leader) - 3인 팀

- 서비스 기획 및 프로젝트 매니저 역할 수행
- 채팅 기반 주문 과정 시스템, 실시간 채팅 알림, 송금 기능 개발
- 사용자 개인 계좌번호를 **AES256 암호화**하여 안전하게 저장
- 카카오페이나 토스와 유사한 **소셜 송금 방식**을 우회적으로 구현
- 채팅방 내 모든 사용자에게 빠르게 알림을 전송하기 위해 **로컬 스토리지 최적화**





Flutter 3.19.3

Naver Map API 1.2.0

Kakao Map API

Firebase

https://tosto.re/hanbab

https://github.com/Han-Bab/HanBab

DETAILS

상세 설명

AES Encryption

문제 정의

- 송금 로직에서 유저의 개인 계좌번호를 입력받아 DB에 저장해야 했음
- **계좌번호는 민감한 개인정보에 해당**
- 유출 위험에 대비해 **보안 강화** 필요성 발생

해결 방법

- **Flutter Encrypt API**를 활용해 16비트 랜덤 문자열 기반의 **AES Encryption**을 적용하여 계좌번호를 안전하게 **암호화 및 복호화**

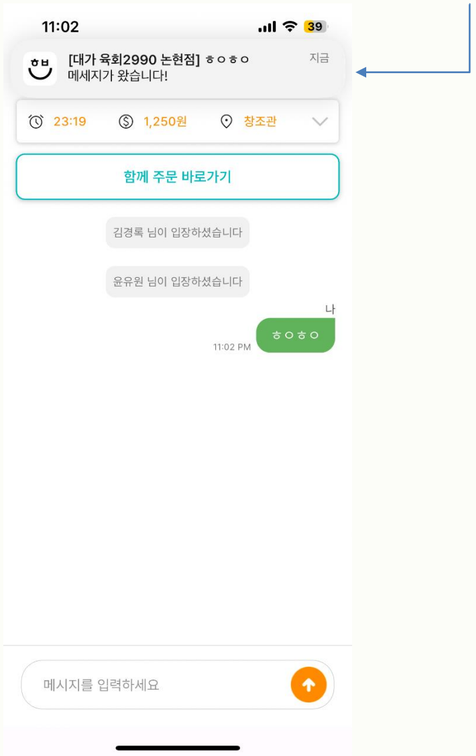
DETAILS

상세 설명

FCM 알림 구현 (Token vs Topic)

문제 정의

- 초기에는 FCM의 **Topic 방식**을 채택해 단체 채팅방에 효율적으로 알림을 전송
- 하지만, **내가 보낸 메시지도 나에게 알림이 전달**되는 구조적 한계 발생



해결 방법

- **Token 방식**으로 전환하되, 성능 문제를 보완하기 위해 **SharedPreferences 기반 로컬 캐싱** 도입
- 단톡방 입장 시 DB에서 토큰 1회 조회 → groupId 기준으로 로컬에 저장
- 이후 메시지 전송 시 **캐시된 토큰 재활용 → DB Read 최소화 및 속도 개선**
- 입장/퇴장 이벤트 시 토큰 목록 실시간 갱신 로직 구현

- **알고리즘 정리 링크**: https://kimgyeonglock.github.io/hanbab/fcm/

구분	**Token** 방식	**Topic** 방식
기본 원리	각 유저가 고유한 token을 갖고, 해당 token으로 개별 전송	pub/sub 방식으로 특정 키(topic)에 구독한 사용자 모두에게 전송
알림 대상 지정	알림을 받을 유저의 token을 직접 조회해 전송	topic을 기준으로 구독한 사용자 모두에게 자동 전송
장점	개별 유저에게 세밀하게 알림 전송 가능	대상자를 직접 지정하지 않아도 되어 구현 간편
단점	**단톡방의 모든 참여자 token을 알아야 함 → 성능 및 관리 부담**	**본인에게도 알림이 오는 구조적 한계 존재**

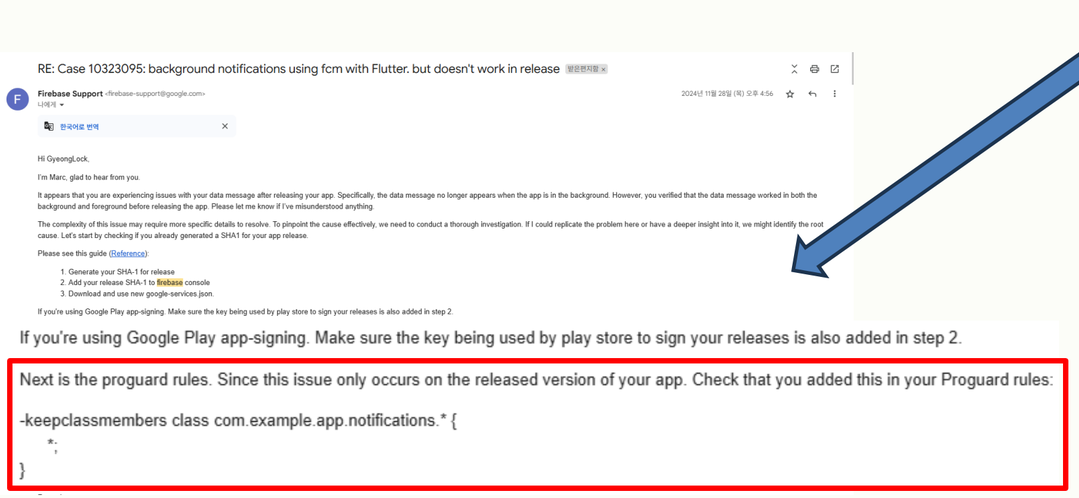
DETAILS

상세 설명

백그라운드 알림 미수신 오류 해결

문제 정의

- 일부 기기에서 **백그라운드 상태에서 FCM 알림이 수신되지 않는 문제** 발생
- 개발 환경에서는 재현되지 않아 **문제 원인 파악 어려움**


RE: Case 10323095: background notifications using fcm with Flutter. but doesn't work in release

Firebase Support <firebase-support@google.com>

2024년 11월 28일 (목) 오후 4:56

Hi GyeongLock,

I'm Marc, glad to hear from you.

It appears that you are experiencing issues with your data message after releasing your app. Specifically, the data message no longer appears when the app is in the background. However, you verified that the data message worked in both the background and foreground before releasing the app. Please let me know if I've misunderstood anything.

The complexity of this issue may require more specific details to resolve. To pinpoint the cause effectively, we need to conduct a thorough investigation. If I could replicate the problem here or have a deeper insight into it, we might identify the root cause. Let's start by checking if you already generated a SHA1 for your app release.

Please see this guide (Reference)

1. Generate your SHA-1 for release
2. Add your release SHA-1 to firebase console
3. Download and use new google-services.json.

If you're using Google Play app-signing. Make sure the key being used by play store to sign your releases is also added in step 2.

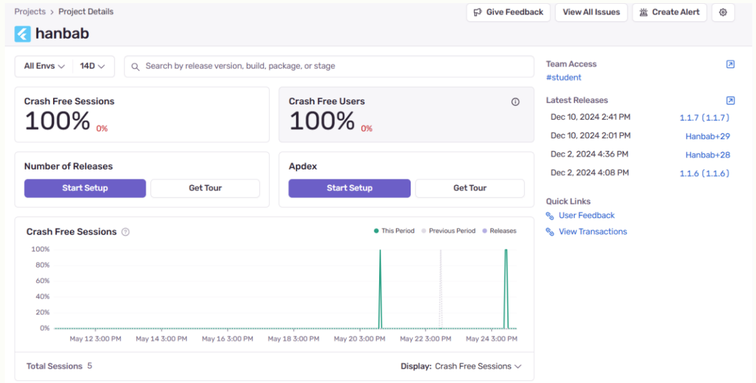
If you're using Google Play app-signing. Make sure the key being used by play store to sign your releases is also added in step 2.

Next is the proguard rules. Since this issue only occurs on the released version of your app. Check that you added this in your Proguard rules:

```
-keepclassmembers class com.example.app.notifications.* {
    *;
}
```

해결 방법

- **Sentry 기반 모니터링 도구**로 실행 로그 추적
- **Firebase Support**에 직접 문의해 원인 분석



원인

Proguard 난독화 설정으로 인해 알림 수신 관련 클래스가 제거됨

해결

Proguard 설정에서 **FCM 관련 클래스 예외 처리**로 코드 보존

김 경 록

Phone

010-8079-5427

Email

kkl5468@gmail.com

Blog

https://kimgyeonglock.github.io