👋

배움을 나누고, 끊임없이 개선하려는 개발자 하동준입니다.

📧 **djh0211@konkuk.ac.kr**

📞 **010-6478-6194**

🐼 Github (https://github.com/djh0211)

LinkedIn (https://linkedin.com/in/djh0211)

🏫 건국대학교 컴퓨터공학부 (졸업) GPA 3.78/4.5

자기 소개

배움을 나누며 함께 성장하는 것을 좋아합니다.

정리한 내용을 바탕으로 **다른 사람들에게 쉽게 설명**해주는 것을 즐깁니다.

현재까지 주 3회 기술 스터디에 참여하며 6개월 동안 **40개 이상**의 발표 자료를 만들었습니다.

발생 가능한 문제점을 고려한 개발을 수행합니다.

타임아웃과 RR스케쥴링을 통해 IoT 기기 간 1:N 블루투스 통신에서 발생할 수 있는 **기아 현상을 해결**하였습니다.

Kafka와 jdk21 가상 스레드를 통해 트래픽 증가 시의 **메시지 유실을 방지**하고 **처리율을 2배 향상**했습니다.

해당 가치관을 바탕으로 삼성 SW 아카데미에서 **3번 중 2번의 프로젝트에서 상을 수상**하는 성과를 이루어냈습니다.

🎓 교육 및 경험

2023.07 ~ 2024.06 (1년)

삼성 SW 아카데미 10기 교육 수료

JAVA 전공반으로서 SW 개발에 관한 실무경험을 쌓기 위해 지원하였습니다.
알고리즘 설계 능력과 3번의 7주 기간 애자일 프로젝트 기획/개발을 통해
웹 및 IoT 개발 능력을 학습하였습니다.

2023.03 ~ 2023.05 (8주)

프로그래머스 주관 데이터 엔지니어링 교육 수료

데이터 직무의 이해와 데이터 팀의 조직 구조 등에 학습하였습니다.
KPI지표를 위한 요약테이블 구성과 데이터 마이그레이션을 위한
SQL 쿼리 작성 능력을 학습하고, S3와 Redshift에서 Airflow를 활용한 **ETL 데이터 파이프라인**을 구축하고, Backfill 능력을 학습하였습니다.

2022.01 ~ 2022.06 (5개월)

네이버 부스트캠프 AI Tech 3기 추천시스템 교육 수료

추천시스템 전공으로서 Machine/Deep Learning 기초 이론과 추천시스템 설계 등을 학습하였습니다. **이미지 분류, 다음 아이템 예측, Knowledge Tracing 도메인**에서 Kaggle 형식의 경진대회를 진행하였습니다. ML 모델 활용 웹서비스 개발 및 배포하였습니다.

2021.03 ~ 2023.08

건국대학교(서울) 컴퓨터공학과 졸업 GPA(3.78/4.5)

2017.03 ~ 2020.12

한국산업기술대학교 컴퓨터공학과 중퇴 GPA(3.46/4.5)

🏆 수상 및 자격

🏆 **2024.05.20**

삼성 SW 아카데미 자율 프로젝트 우수상(11팀 중 2등), 전국 본선 진출

IoT 냉장고와 스마트 용기를 통한 신선도 관리 시스템 - BeFresh

🏆 **2024.04.05**

삼성 SW 아카데미 특화 프로젝트 우수상(11팀 중 2등), 전국 본선 진출

AI 기반 미술 심리 검사를 통한 심리 치료 서비스

📄 **2023.09.14**

OPIc(영어) IM2 취득

📄 **2022.09.30**

SQLD 자격증 취득

프로그래밍 역량

Back-End

DB

- 다수의 RDB 설계 경험 (InnoDB/OracleDB)
- **인덱스 전략 및 쿼리 설계**
- NoSQL(Mongo), Cache(Redis), 시계열DB(InfluxDB) 활용 경험
- Redshift/BigQuery 등 DW 활용 및 Airflow 통한 **데이터 마이그레이션** 역량

Spring Boot

- 4개의 프로젝트에서 사용
- JPA, QueryDSL, Mongo Template을 통한 REST API 개발 경험

Spring Cloud

- MSA 기반 프로젝트 개발 경험
- MSA의 사용 이유와 기본적인 아키텍쳐 구성에 대한 이해

언어

JAVA

- 알고리즘 및 대다수 프로젝트의 주요 언어
- JVM 동작 원리 및 jdk21 가상스레드에 대한 이해

Python

- **Asyncio/Multiprocessing** 활용
- 데이터 ETL작업에 활용
- ML 라이브러리 활용

Cloud/Infra

- **GCP, AWS, Docker, Nginx, Jenkins**
  - Git과 연동한 CI/CD 구축
  - blue-green 무중단 배포 구현

Collaboration

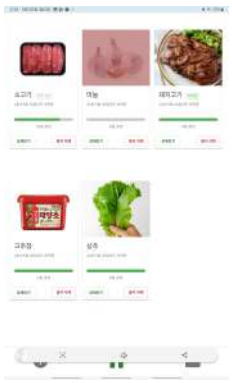
- **Git, Jira, SonarQube**
  - 애자일 기반 프로젝트 진행
  - git-flow 전략
  - 코드 분석을 통한 품질 향상

프로젝트

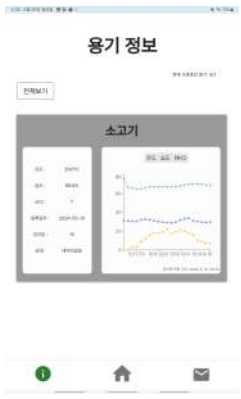
Be Fresh

IoT 냉장고(라즈베리)와 스마트 용기(아두이노)를 통한 신선도 관리 시스템

> 💡 **[프로젝트 개요]**
>
> 냉장고 내부의 음식이 있는지 잊어버리거나, 음식의 신선도를 정확히 파악하기 어려운 문제를 해결하기 위해 **식자재 신선도 관리 플랫폼**, **비프레시**를 만들었습니다.
>
> 아래와 같은 서비스를 제공합니다.
> 1. 스마트 용기, 유통 식품, 일반 음식을 냉장고에 등록하여 관리 가능
> 2. 음식의 신선도와 관련된 다양한 정보를 확인 가능
> 3. 식자재 관련 알림 제공



냉장고 음식 관리 화면



용기 센서 데이터 그래프

실제 제품 사진

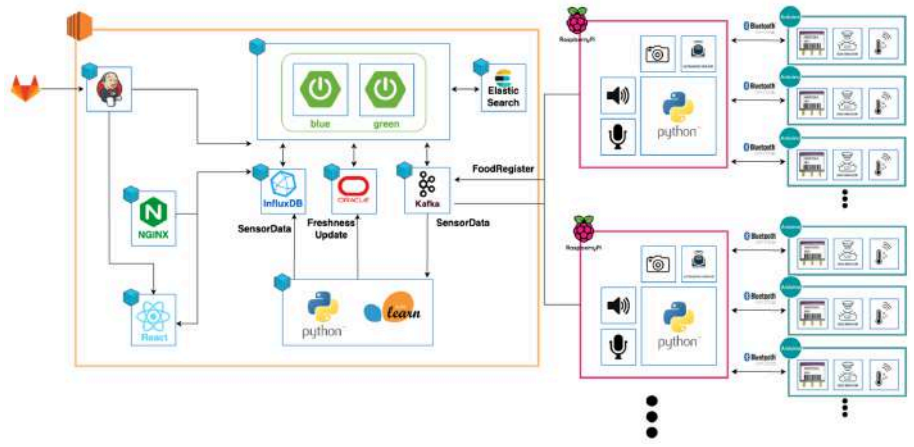
기간 2024.04.08 ~ 2024.05.20 (7주)

팀 6명 **역할** IoT/Backend **기여도** 25% **수상** 삼성SW아카데미 최종 PJT **우수상(11팀 중 2등)**

깃허브 github.com/f17coders/be-fresh

참고 자료

프로젝트 발표 자료

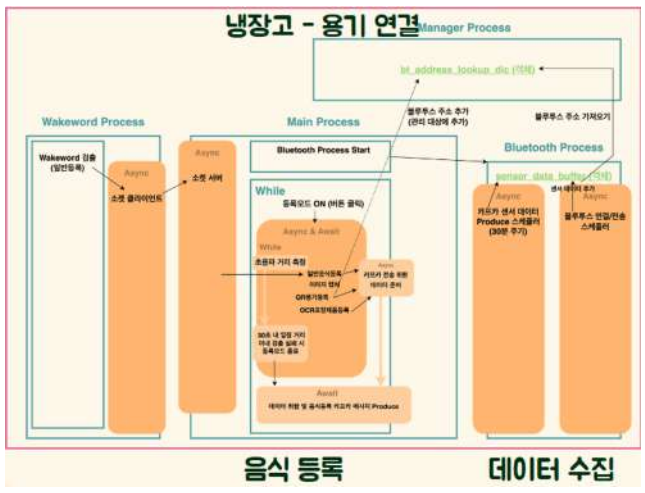


전체 아키텍처

개인 기여 부분

냉장고의 관리 모듈 개발

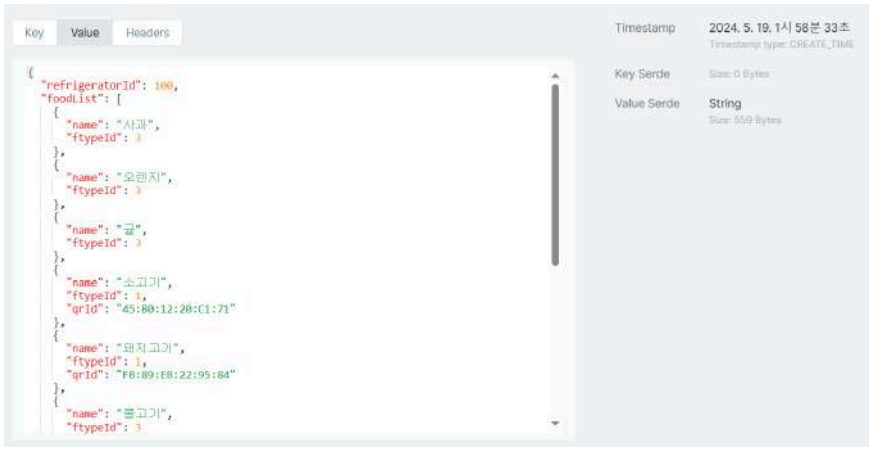
- GPIO 활용 버튼클릭 모듈(등록 모드 활성화)
  - 활성화 시에만 다른 모듈 작동 → 전원 절약
- GPIO 활용 초음파센서 거리 측정 모듈(자리 이탈 감지 및 등록 모드 자동 종료)
- Picamera2, OpenCV 활용 카메라 촬영 모듈
- pyzbar 활용 QR코드 검출 모듈(용기에 부착된 QR코드 인식하여 해당 용기의 블루투스 주소 획득)
- pytesseract 활용 OCR 검출 모듈(유통 식품의 유통기한 검출)
  - 정규 표현 식 활용하여 (YY/YYYY).(M/MM).(d/dd) 형태만 검출
- Picovoice 활용 wakeword 검출 모듈(사용자가 '일반 등록' 이라고 말하면 위 두 종류 외의 일반 음식 등록 가능)
- SpeechRecognition 활용 STT 모듈(사용자에게 해당 음식의 이름을 제공 받음)



냉장고 관리 모듈 구조

냉장고 관리 모듈 비동기 병렬 통합

- 동기 진행 시, 대기 시간이 길어지기 때문에 IO작업은 **Asyncio**를,
  CPU는
  Multiprocessing을 통해 최적화
- Kafka를 통한 음식 등록과 센서 데이터 주기 전송 또한 데이터를 하나로 취합하여 비동기 작업으로 구성 →
  대기 시간 및 요청 수 최적화



🖥️ 사용 기술

- `Spring Boot 3.2` : **첫째** 코드 수정 등 유지보수 용이성, **둘째** Client에 제공할 API 개발, Kafka, ElasticSearch 등 여러 프레임워크의 대응을 위해 냉장고-Client의 직접 연결 보다 중계 서버를 추가했습니다. 또한 jdk21 활용을 위해 **v3.2** 를 선택했습니다.
- `Kafka` : RestAPI를 사용하면 대규모 트래픽 증가시 **타임 아웃 및 메시지 유실**이 가능합니다. Kafka를 도입하면 수신자의 처리 속도에 맞게 메시지 수신 가능하고, 오프셋 수동 커밋을 통해 실패 메시지 별도 관리가 가능합니다.
- `InfluxDB` : 센서 데이터에 대한 쓰기 작업과 쿼리 부하를 처리하기 위해 도입하였습니다.
- `Python 3.8` : 카메라 및 센서 코드 작성을 위해 사용했습니다. 또한 녹음 관련 라이브러리인 PyAudio가 3.9 이상을 지원하지 않았고, Asyncio 활용을 위해 3.8버전을 선택했습니다.
- `Asyncio` : 통신 및 sleep 등의 **IO 작업으로 인한 블로킹을 개선**하고, 모듈의 비동기 반복 실행을 위해 도입하였습니다.
- `Multiprocessing` : 지속적인 CPU작업과 **멀티 코어 활용**을 위해 도입하였습니다.

스마트 용기의 센서 데이터 BLE 통신

- Multiprocessing의 Process 활용하여 **별도의 프로세스**로 구성
- 하나의 냉장고가 여러 개의 스마트 용기와 **주기적, 순차적 블루투스 통신**을 하기 위해 **스케줄링**을 적용

스마트 용기 신선도 업데이트 관련 배치 작업

- **Kafka** 센서 데이터 토픽의 컨슈머 코드 작성, 메모리에 센서 데이터를 적재
- 센서 데이터에 대한 **쓰기 작업과 쿼리 부하를 처리**하기 위해 시계열 데이터베이스 **InfluxDB** 도입
  - 용기 → 냉장고 → 파이썬 서버 → InfluxDB 순으로 전달된 센서 데이터를 Client에게 전달

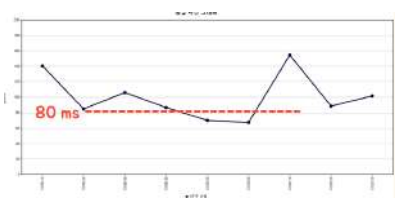
- 30분 주기 배치 작업
  - 데이터 csv화
  - InfluxDB에 적재
  - 30분 이내의 센서 데이터가 존재하는 용기의 경우 ML 모델 통해 신선도 업데이트
  - Oracle DB에 신선도 값 업데이트

음식 등록 과정 비동기 처리

- 다수의 요청에 대해 병렬 처리하기 위함

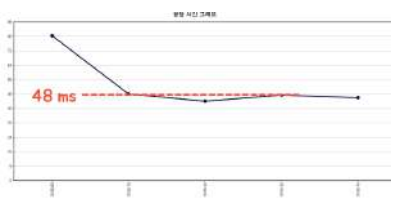
  ⇒ **가상 스레드 적용** : 플랫폼 스레드 보다 생성 비용이 작고, 논블로킹 방식
- 1000 개의 요청에 대한 응답 시간 및 처리율 비교

[플랫폼 스레드]



- 처리율 : 10.9/sec

[가상 스레드]



- 처리율: **20.2/sec**

⇒ **플랫폼 스레드에 비해 더 높은 처리량과 빠른 응답시간**

제품 업데이트 대비 + 냉장고의 관리 모듈 시작 프로그램 화

- `crontab` 활용하여 1주일 주기로 최신 코드 git pull 하는 스크립트 실행
- /etc/xdg/lxsession/LXDE-pi/autostart 에 파이썬 스크립트 실행 명령어 추가

트러블 슈팅

🚨 wakeword 검출 모듈과 Picamera2 모듈이 Asyncio와 Multiprocessing 환경 모두에서 정상적으로 **작동하지 않았습니다.**

⭐ **해결 방법**

- wakeword 검출 모듈을 독립적인 실행 지점을 갖는 파일로 구성하고 카메라 모듈(메인 프로세스)와 **IPC통신**
- Asyncio의 **소켓 통신**을 이용하여 wakeword 검출 모듈(Client)이 카메라 모듈(Server)에 검출을 전달
- Client는 매 전달 때마다 최대 4회 연결을 시도하고, Server는 예외 처리를 통해 **비정상 종료 시 서버 Open을 계속 시도**

🙌 **결과**

- 무한 동작하는 wakeword 검출 모듈을 **메인 프로세스에서 분리하여 성능적 이점과 함께 결합도 감소**

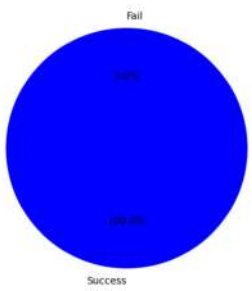
📚 **배운점**

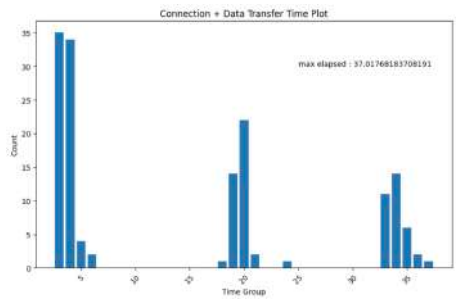
- Python은 **멀티 스레드**를 활용하여도 GIL에 의해 결국 **하나의 프로그램 카운터**로 동작한다.
- 특정 라이브러리의 경우 Asyncio 혹은 Multiprocessing에 안정성을 갖지 않으므로, **개발 전 확인**이 필수적이다.

🚨 냉장고 라즈베리가 동시에 2개 이상의 아두이노와 연결 시도 및 연결을 유지하려 할 경우 에러가 발생(**블루투스 버전으로 인해 동시 작업이 불가**)

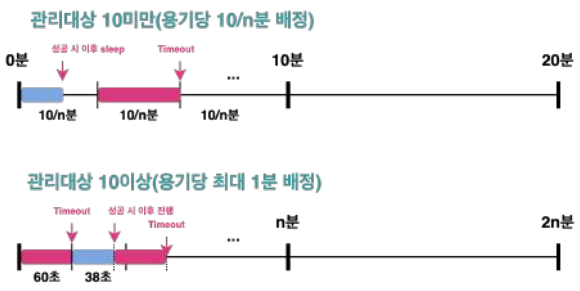
⭐ **해결 방법**

- 블루투스 작업 논블로킹 비동기 실행 미 적용
- 연결 유지 → 연결 + 데이터 송수신 + 연결 해제
- 또한, 개별 용기마다 주기마다 충분한 연결 시도를 위한 시간 할당 필요(**기아 문제 방지**)
- **60분 간** Serial하게 2m간격의 2대의 아두이노와 BLE통신/데이터 수신을 반복(최대 4번의 연결 시도 허용)하는 **테스트 실행**





  - 최종 연결 100% 성공
  - 50% 5초 미만 소요, 10% 약 35초 ~ 최대 37초 소요 → **최대 타임 아웃 60초** 설정
  - ML 모델의 학습 데이터 간격 10분 → 기본 연결 주기 10분 설정
- **RoundRobin 알고리즘(시분할)** 적용



  - 10분/n = 1분이 되는 n=10을 기준으로 로직 자동 변경을 적용
  - 최악의 상황에도 용기당 **n분 주기로 60초의 연결 시도 가능**

🙌 **결과**

- 무한 동작하는 wakeword 검출 모듈을 메인 프로세스에서 분리하여 **성능적 이점과 함께 결합도 감소**

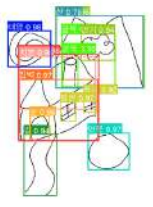
📚 **배운점**

- 작은 문제에서는 발생하지 않았던 현상이 큰 문제(업소용)에서는 발생한다. 따라서 **기저를 확장해서 문제점을 찾아보는 습관**을 기르자.

나여
나를 위한 여정

AI 기반 미술 심리 검사를 통한 심리 치료 서비스


HTP 검사란?


YOLO v5 객체 탐지


HTP 검사 결과


메인 화면


데일리미션 도메인


데일리미션 도메인

> 💡 **[프로젝트 개요]**
>
> 2022년 청년 삶 실태조사 결과, 최근 1년 번아웃 경험 34%, 우울증 경험 6.1%로 청년층이 심리적으로 어려움을 호소하지만, 이에 대해 도움을 청할 사람이 없다는 문제점을 인식하여 다음과 같은 서비스를 기획하였습니다.
>
> **AI 기반의 HTP 검사**를 통해 사용자의 심리상태를 진단 후 8가지 유형으로 분류합니다.
> 해당 유형에 따라 다음과 같은 서비스를 제공합니다.
>
> 1. 유형별 데일리 미션 추천
> 2. 날마다 다른 질문에 대한 생각을 공유하는 **포스트잇 커뮤니티**
> 3. 고민을 이야기하고 들어주는 고민 상담소

기간 2024.02.26 ~ 2024.04.05 (7주)

팀 6명 **역할** Backend **기여도** 20% **수상** 삼성SW아카데미 특화 PJT **우수상(11팀 중 2등)**

깃허브 github.com/f17coders/mindtrip

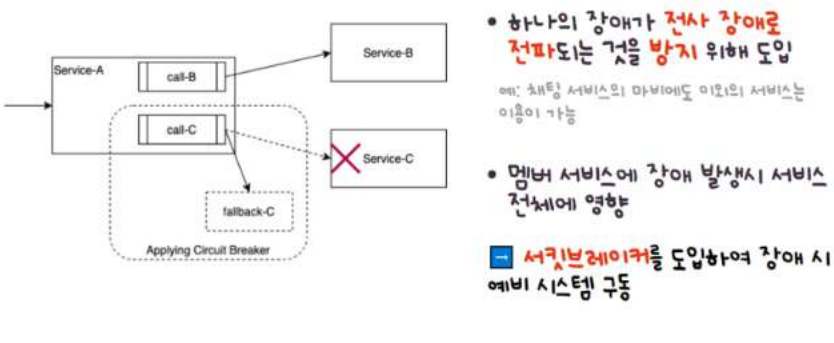
참고 자료

> 프로젝트 발표 자료

개인 기여 부분

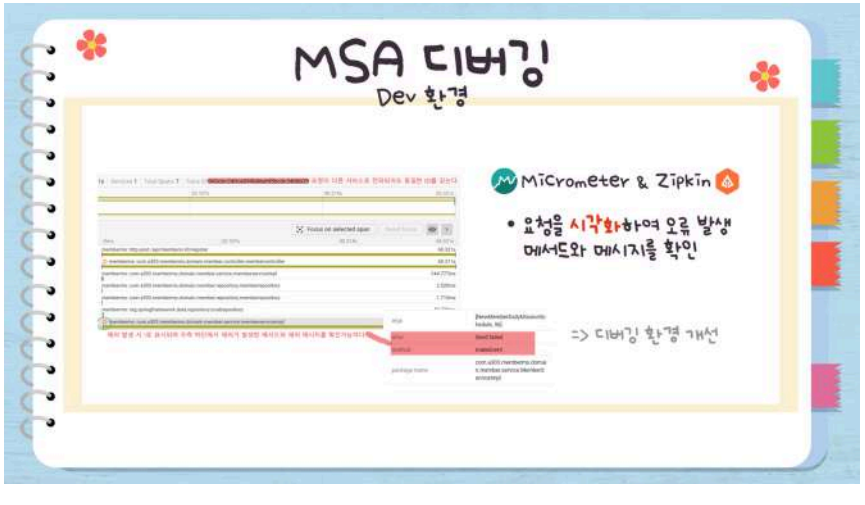
MSA 기반 아키텍처 설계 및 구축

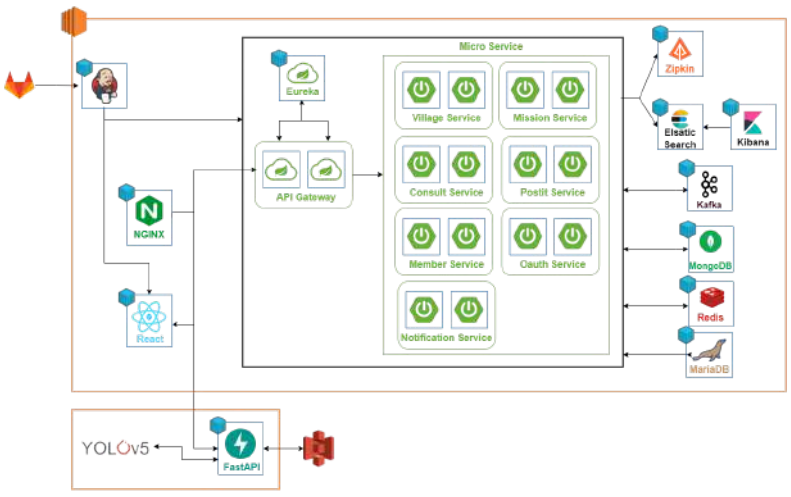
- 하나의 장애가 전사 장애로 전파되는 것을 방지하기 위해 도입
- 서비스 분할 및 DB **설계**(MariaDB, MongoDB)
- Spring Cloud Framework 활용
- 멤버 서비스(서비스들과 동기적으로 연관)의 장애는 전사 장애로 전파
  →
  서킷브레이커를 도입하여 장애가 생겼을 때 스케쥴링된 읽기용 멤버 DB를 활용하여 데이터를 정상적으로 반환하도록 하였습니다.



서버 디버깅 환경 구축

- 분리된 서비스로 인해 **서버에서의 디버깅이 어려움**
- **Micrometer**를 활용하여 하나의 요청에서 전파되었을 때에도 같은 traceId로 관리
- **Zipkin**을 활용하여 요청을 시각화하고, **오류 발생 메서드 및 메시지 확인**
- **ELK**를 활용하여 최근 2시간 발생 **에러 대시보드** 구축





서비스 아키텍처

🖥️ 사용 기술

- `Spring Boot 3.2` , `Spring Cloud` , `Spring Data JPA`
- `Kafka` : 이벤트 기반 아키텍쳐 구성과 알림 시스템 구축을 위해 도입하였습니다.
- `ELK` : 서비스 무관하게 로그의 형식을 통일하고, Kibana에서의 활용을 위해 LogStash를 사용하였습니다.
- `FCM` : 엔드포인트 유형 무관하게 알림을 발송하고, 발송 및 커넥션 관리를 외부 위임하여 메모리 사용을 줄이기 위해 도입하였습니다.

트러블 슈팅

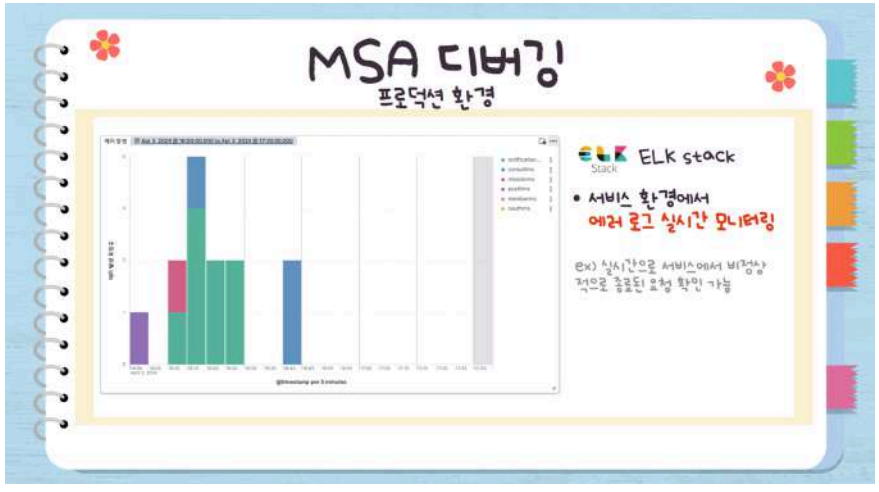
> 🧯 daily_mission(오늘 수행하고자 하는 미션들)의 경우 매일 12시 기존 테이블을 mission_log(오늘 이전의 미션 수행 기록)에 추가하고, 전체 유저에게 3개의 미션을 추천하여 다시 넣어준다.
>
> **→ save/delete 과정에 사용되는 세션 개수 및 쿼리의 개수 최적화 필요**

⭐ 해결 방법

- Spring Data JPA saveAll/deleteAll 메서드는 DB세션을 한번만 열기 때문에 for문을 통한 save/delete 메서드보다 빠르다.
- 하지만 이 또한 개별 insert/delete 쿼리를 여러 번 실행한다는 단점 존재
- 따라서 JDBC Template을 활용하여 벌크 CUD쿼리를 최적화

📚 배운점

- Spring Data JPA를 통해 작성된 쿼리가 RDBMS 상에서 **기대하는 것과 동일하게 동작하지 않을 수 있다.** 개발 중에는 로그를 **디버깅 모드로 전환**하여 실제 어떻게 동작하는지 살펴보자.



통합 알림 시스템 구축(ver1: SSE+Kafka ver2: FCM+Kafka)

- 각 서비스에서 Kafka에 메시지 발행
- 알림 서비스에서 메시지 수신하여 DB에 기록하고 알림을 Client에 발송
- 초기 SSE를 활용하였으나, 메모리 관리 위하여 **FCM** 도입
  - 발송 및 커넥션 관리를 **외부 위임하여 메모리 관리 용이**

데일리미션 도메인 API 개발

- Spring Data JPA, QueryDSL 활용
- Spring Boot Scheduler 활용
- JDBC Template 활용하여 Bulk CUD 쿼리 최적화

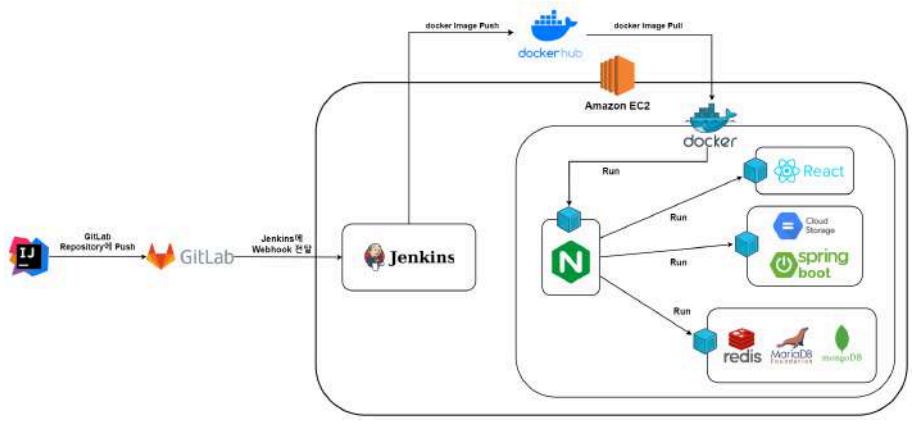
CLASSHUB

온라인 IT 강의 통합 플랫폼

💡 **[프로젝트 개요]**
온라인 강의의 수요가 높아짐에 따라 수많은 IT 강의 플랫폼이 생겨 **정보의 파편화**가 발생하였습니다. 또한, 학습자들이 스터디 그룹을 구성하거나 **학습 경험을 공유하는 공간이 부족**하여 사용자들에게 불편함을 주고 있어 사용자들에게 편의성과 효율성을 제공하는 서비스를 개발하고자 **온라인 IT 강의 통합 플랫폼**을 기획하였습니다.

해당 서비스에서는 다음과 같은 서비스를 제공합니다.

1. IT 강의 사이트들의 강의 정보 및 리뷰 제공
2. 강의를 한 눈에 비교
3. 커뮤니티
4. 스터디 구성 기능



서비스 아키텍처

기간 2024.01.08 ~ 2024.02.16 (7주)

팀 6명 **역할** Backend/Data **기여도** 20%

깃허브 github.com/f17coders/ClassHub

기여 부분

강의 데이터 ETL 작업

- 유데미, 인프런, 구름에듀 7만건의 강의 메타데이터, 900만건의 리뷰 스크래핑
  - 5대의 GCP VM 사용 → 로우 데이터 GCS 적재
- **통합 데이터 모델링**
  - 카테고리(Not Null)

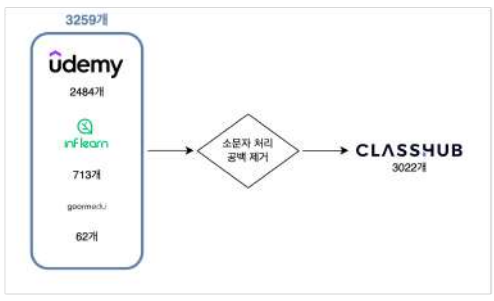
  - **태그 테이블 도메인 설정**(Nullable & 강의와 N:M)
    - **소문자 처리 + 공백 제거**했을때 동일하다면 일치


랜딩페이지 강의검색 강의상세페이지

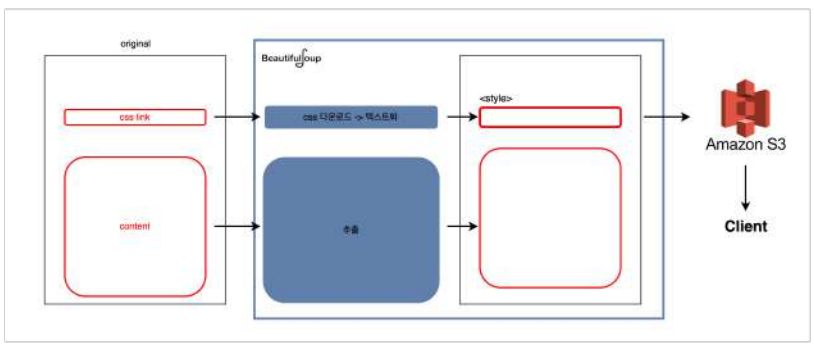
🖥️ 사용 기술

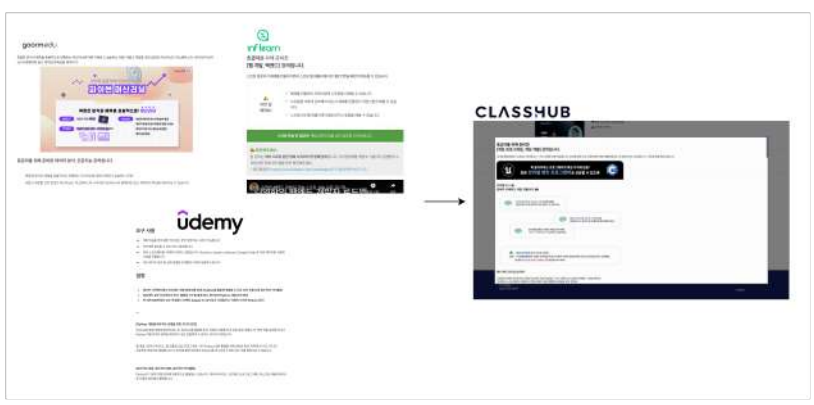
- `Spring Boot 3.2` , `Spring Data JPA` , `MariaDB`
- `Python` , `Pandas` , `BeautifulSoup` , `SQLAlchemy`
- `Google Cloud Platform`

- 만일 새로 추가되는 강의가 처음 보는 태그를 갖는다면
  →
  태그 자동 추가 후 강의 추가



- **강의 소개 페이지**
  - 정해진 형식 없이 css 적용된 html 페이지로 제공됨
  - **CSS 링크로부터 텍스트로 다운 + content 추출** → 새로운 html 문서, S3 저장 → Client 제공





강의 및 리뷰 도메인 API 개발

- Spring Data JPA + QueryDSL 활용 동적 쿼리 개발
- 강의 통합 별점 지표 개발
  - **서비스의 리뷰 별점 Cold Start**를 해결하기 위해 **가중치 부여**

combinedRating: 실제 강의 사이트와 우리 서비스의 리뷰 별점을 통합하며 서비스 상에 표시되는 별점입니다.

우리 서비스의 리뷰 별점 개수	우리서비스 별점 가중치	외부 사이트 별점 가중치
10개 미만	0.2	0.8
10개 이상	0.5	0.5

- **추천순 지표 개발**

combinedRatingCount: 실제 강의 사이트 리뷰 개수 + 우리 서비스의 리뷰 개수

lectureLikeCount: 우리 서비스의 사용자들이 해당 강의를 찜한 개수

weight: 추천순 정렬을 위해 계산되는 값입니다.

```
실제 강의 사이트 수강생 수 * 0.4 + combined_rating * 0.35 +
lecture_like_count * 0.15 + combined_rating_count * 0.1
```

LLM 리뷰 요약

- 수집된 리뷰들을 Google Palm2 를 활용하여 강의의 좋은 점과 아쉬운 점으로 요약 (7만 건 대상 약 3만원 사용)
- 아쉬운 점: **감성 분류 ML 모델**을 활용하여 긍정/부정 리뷰로 분류하고, 그 신뢰도가 높은 리뷰 위주로 Palm2에 제공했다면 토큰을 줄여 **가격을 줄이고 더 좋은 결과**를 얻을 수 있을 것 같다.

트러블 슈팅

약 7만개의 강의에 대해 필터링, 정렬, 페이징 등을 수행하는 데 많은 시간 소요

- lecture, lecture_like, lecture_buy, category, lecture_tag, tag 와의 Join 필요
- lecture 테이블의 칼럼들 외에 별점 등 실시간 계산되어 서비스에 사용되는 값들이 다수 존재하였습니다.

⭐ **해결 방법**

- ~~Redis 캐싱 및 주기 업데이트?~~
  - 해당 값들을 활용하여 정렬 및 페이징 필요, **Redis** 사용 시 요청 마다 Spring Server에서 **CPU 활용** 수동 결합 요구
- ~~테이블 역 정규화 등한 해당 칼럼 추가?~~
  - Read 작업과 Update 작업이 **혼재**
- Join의 편리성을 위해 Redis가 아닌 **별도의 요약 테이블**(lecture_summary) 을 두고 1시간에 한번씩 스케쥴링을 통해 테이블을 갱신(delete/insert into select)
  - **Read Only** 강의 테이블

🙌 **결과**

(7만건의 강의 대상) 기능별 API 전송 속도

description	속도(단위: ms)
별점순으로 강의 정렬 후 1페이지 쿼리	122
추천순으로 강의 정렬 후 1페이지 쿼리	133
가격순으로 강의 정렬 후 1페이지 쿼리	88
태그 포함 모든 필터링 요소 적용 후 1페이지 쿼리	1426
필터링 미 적용 후 검색어 기반 쿼리	1266
필터링 적용 후 검색어 기반 쿼리	1602

📚 **배운점**

- API 전송 속도 측정 대신 **부하테스트 툴**을 사용해서 체크하자.

강의명, 강사명 부분 검색 기능 성능 부족

- Spring Data JPA 및 QueryDSL: contains 키워드(`like %word%`)
  - 부분 검색 지원하나 인덱스 활용 불가 → 낮은 성능
- `like %word` 혹은 `like word%`
  - 인덱스 활용 가능하나, 시작 지점 혹은 종료 지점만 검사

⭐ **해결 방법**

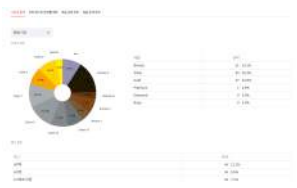
- MariaDB: match() against(.. in boolean mode) 조건을 통해 Full-Text Index 를 활용한 Full-Text Search 를 지원
- Spring Data JPA/QueryDSL 은 **해당 메서드를 지원 X**
  - FunctionContributor 를 extends하고 match() against(.. in boolean mode)를 **사용자 정의 함수**로 생성하여 **인덱스**를 활용한 효율적인 검색이 가능

📚 **배운점**

- 인덱스를 탈 것이라 기대한 쿼리가 그렇지 않을 수 있다. 항상 **explain 키워드**를 통해 확인하자.

AI 기반 코딩 테스트 문제 개인화 추천 서비스

<코딩테스트 어드바이저의 로그인 화면>



<코딩 테스트 어드바이저의 사용자 통계 화면>

<코딩 테스트 어드바이저의 사용자 통계 화면 내 학습이력 그래프의 모습>

<코딩 테스트 어드바이저의 복습 문제 추천 화면>

<코딩 테스트 어드바이저의 예습 문제 추천 화면>

<코딩테스트 어드바이저의 코딩 테스트 합격률 예측에서 하위 분류를 통해 코딩 테스트를 선택한 모습>

> 💡 **[프로젝트 개요]**
>
> 코딩 테스트가 갖는 특수성과 기존 시스템의 한계를 바탕으로, 코딩 테스트를 효율적으로 학습할 수 있게 하는 `코딩 테스트 어드바이저`를 만들고자 한다. 우리는 이번 프로젝트에서 `코딩 테스트 어드바이저`를 통해 크게 4가지 목표를 이루고자 한다.
>
> - 사용자의 실력을 파악하고 실력 향상을 위한 효율적인 학습을 제시한다.
> - 이전에 실시된 기업 코딩 테스트를 사용자가 풀어보지 않아도 **예상 정답률을 예측**해주는 서비스를 제공한다.
> - 일반적인 통계 모델이 아닌 딥 러닝 모델을 사용해 기존 통계 모델의 한계를 극복하고, 이를 통해 사람의 학습을 시간의 흐름에 따라 복합적으로 이해한다.
> - 실시간 딥 러닝 모델 인퍼런스를 위한 클라우드 인프라를 구축한다.

기간 2023.02 ~ 2023.05 (3달)

팀 3명 **역할** 전체 개발 **기여도** 60%

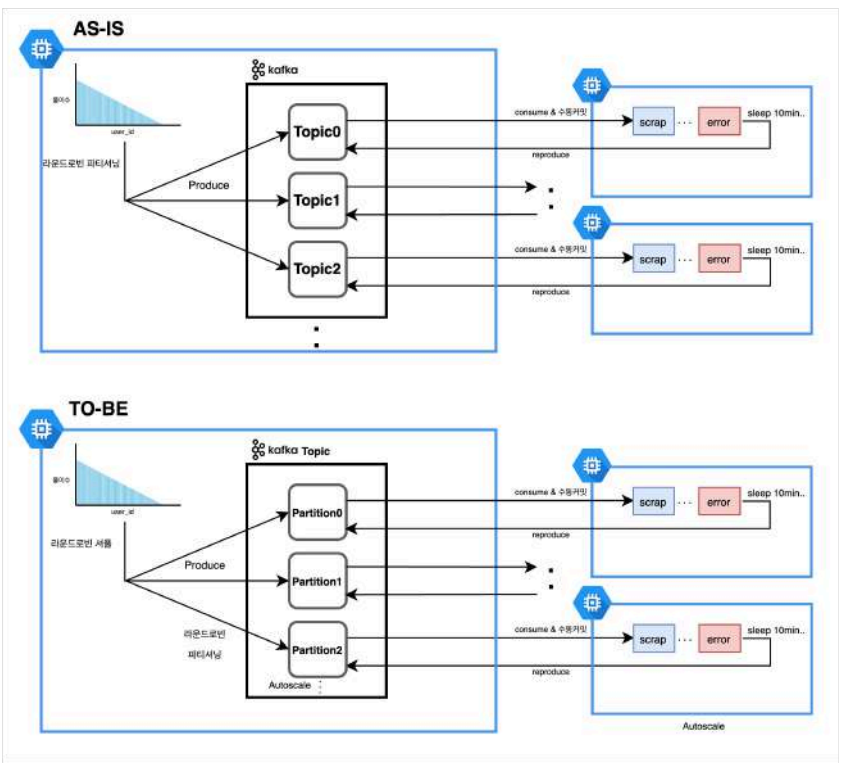
깃허브 github.com/djh0211/CodingTestAdvisorService

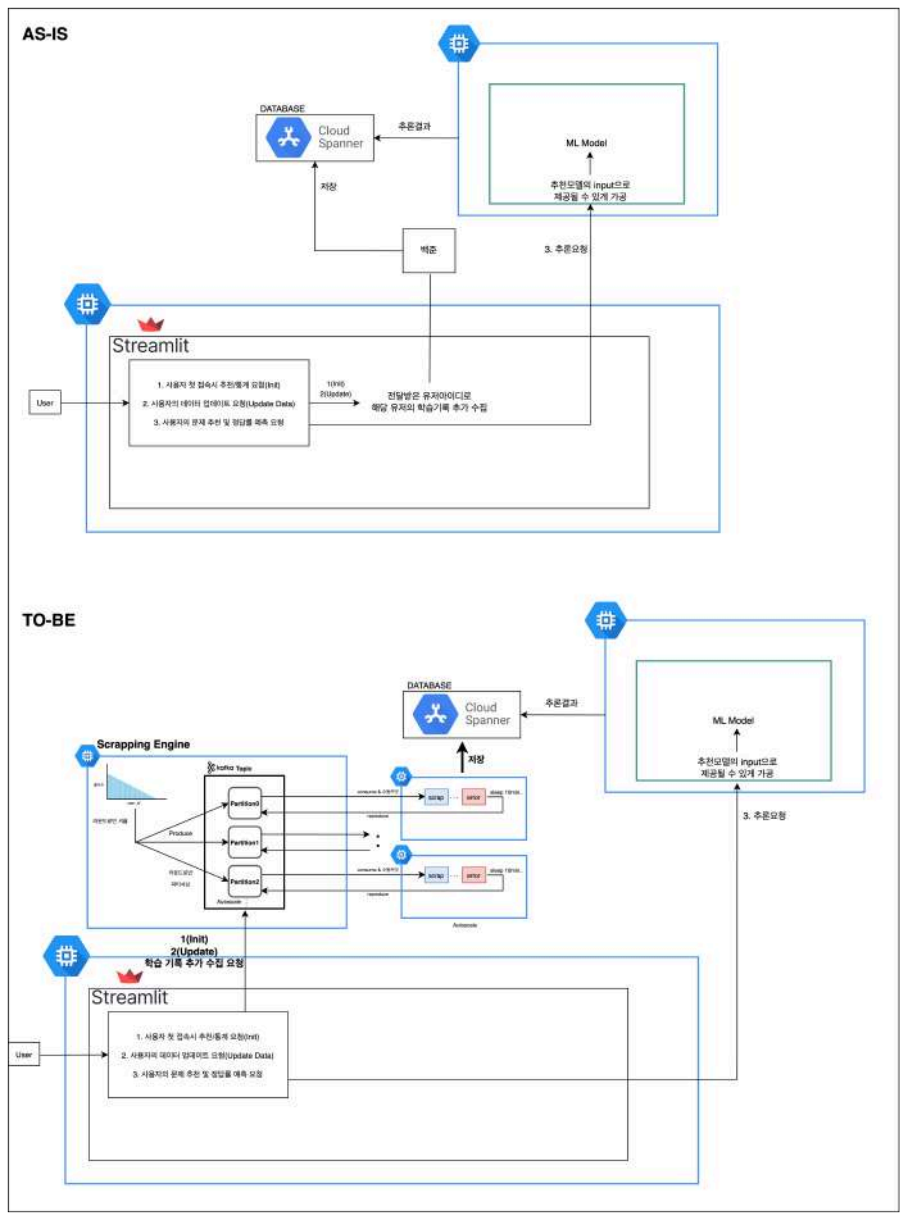
참고 자료 최종 보고서

기여 부분

백준 문제 풀이 이력 데이터 ETL 작업

- 2,500만 건의 문제 풀이 이력, 5만 건의 사용자 정보, 2만5천 건의 문제 정보 스크래핑
  - Kafka 활용 **마스터 슬레이브 패턴**의 스크래핑 시스템 구현
  - **아쉬운 점** : 하나의 토픽에 **라운드로빈 파티셔닝**을 통해 데이터를 분배하고, **컨슈머그룹**으로 데이터를 소비했다면, **Auto Scaling**이 가능했을 것





아쉬운 점 : 웹 서버에서 Kafka로 학습기록 추가 수집 요청을 발행하면 웹 서버의 부담을 덜 수 있었을 것

- 프로덕션 DB를 위한 데이터 가공 작업
  - 스크래핑 - Google Cloud Storage - BigQuery - **Spanner**(프로덕션 DB)
  - Spanner
    - 기존 RDB에 비해 **수평적 확장성**에 가능하여 대규모 트랜잭션 처리 및 **대용량 데이터 저장**에 용이
    - RDB와 유사한 **Google SQL을 제공**하며, 완전한 ACID **트랜잭션** 제공
    - 무엇보다 10GB 미만의 경우 **무료로 제공**

🖥️ 사용 기술

- `Streamlit`
- `Python`, `Pandas`, `BeautifulSoup`, `Sklearn`
- `Google Cloud Platform (GCS, GCE, BigQuery, Spanner)`

쿼리 또는 요청 태그	CPU 사용률	CPU(%)	실행 횟수	평균 지연 시간(밀리초)	스캔된 행의 평균 개수
WITH my_history AS (SELECT * FROM lear...		64.61	335	412.15	49,964.76
WITH tmp1 AS (SELECT A.*,B.title,B.level_i...		6.23	24	488.86	124,406.46
WITH tmp1 AS (SELECT A.*,B.title,B.level_i...		5.69	21	474.6	123,984.71
WITH my_history AS (SELECT * FROM lear...		4.36	403	39.06	25,781.5
WITH tmp1 AS (SELECT A.*,B.title,B.level_i...		1.29	5	399.9	124,586.8
-- SELECT * FROM question TABLESAMPL...		1.2	3	749.59	127,149
SELECT submitted_epochtime FROM lear...		1.15	41	143.17	5,099.46
SELECT * FROM learning_history WHERE ...		0.83	122	14.85	494.46

복잡한 쿼리의 경우에도 최대 1초 이내 제공중

모델 학습 및 배포

- 시간의 흐름을 반영한 **Feature Engineering** 적용하여 모델 학습에 활용(Pandas)

feature name	Description
lag_time	현재 row와 사용자의 직전 학습기록 간의 시간 간격(초)
question_lag_time	현재 row의 문제를 사용자가 얼마만에 다시 푸는지 시간 간격(초)
accuracy_per_question	전체 유저 대상 문제의 평균 정답율(0~100 사이 정수 값으로 변경)
tag	한 문제는 여러 알고리즘 태그를 갖는다. 문제들이 갖는 모든 태그 조합을 1~ 의 정수로 나타내고 부여된 태그가 없는 경우 0을 부여
tag_lag_time	유저가 현재 row 이전에 해당 태그를 얼마만에 다시 푸는지 시간 간격(초)
accuracy_per_tag	전체 유저 대상 해당 태그의 평균 정답율(0~100 사이 정수 값으로 변경)
cumulated_points	result: 문제를 맞췄을때 1 틀렸을때 0 relative result: (100*result) - (accuracy_per_question) cumulated_points: relative result를 누적한 값으로 시점에 유저의 상대적인 실력을 나타냄(정수 값으로 변경)
prior_correct_count	유저가 해당 로우 이전까지 몇번의 맞은 학습기록이 있는지
prior_count	유저가 해당 로우 이전까지 몇번의 학습기록이 있는지
prior_accuracy	유저의 해당 로우 이전까지의 정답율
prior_tag_correct_count	유저가 해당 로우 이전까지 해당 태그를 가진 문제에 대해 몇번의 맞은 학습기록이 있는지
prior_tag_count	유저가 해당 로우 이전까지 해당 태그를 가진 문제에 대해 몇번의 학습기록이 있는지
prior_tag_accuracy	유저의 해당 로우 이전까지 해당 태그를 가진 문제에 대한 정답율

< LightGBM 모델 학습에 사용한 feature>

- 사용자 - 문제 정답률 예측 모델
  - LightGBM
  - 소요되는 학습시간이 짧고 고사양의 컴퓨팅 리소스를 사용하지 않더라도 대규모 데이터 셋에 효율적
  - Feature Engineering을 통해 생성한 다양한 feature 학습에 사용 가능

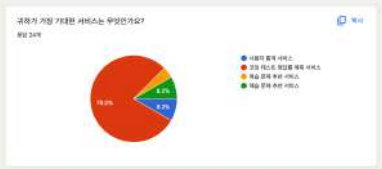
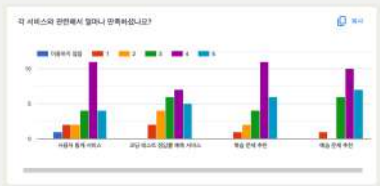
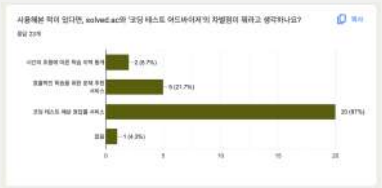
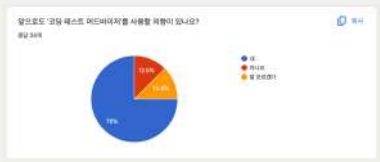
training binary_logloss = 0.53	validation binary_logloss = 0.54
validation AUC = 0.78	validation ACC = 0.72
test AUC = 0.79	test ACC = 0.73

웹 서비스 개발

- 당시 웹 개발 경험 부족으로 FrontEnd와 BackEnd 모두를 하나의 프레임워크로 제공하는 Streamlit 사용

트러블 슈팅

서비스 사용성 조사를 통한 피드백

- 이외 서비스에 대한 총평
  - “전반적으로 어떤 기능을 사용하든 **반응 속도가 너무 느려** 계속 이용하고 싶은 의지가 매우 낮아졌습니다. 다른 것보다도 우선 이 로딩 속도 개선(혹은 캐싱)이 필요할 것 같습니다. 뭘 누를 때마다 로딩이 걸리는 것이 불편합니다.”

⭐ **해결 방법**

- DB 쿼리 성능은 전체적으로 1초 이내인 반면, **웹 서버에서 부가적인 조립 및 데이터 전처리**가 이루어지는 문제 발생
  - **ORM 기술 활용**
  - DB 역 정규화 고려
- 사용자의 버튼 입력을 통한 데이터 업데이트 요청(스크래핑 작업)이 **서버**에서 이루어진다.
  - 초기 데이터 수집 과정에서 만들었던 스크래핑 엔진을 활용하여 외부에 **비동기 요청**
- ML 추론 API에 **메세지 큐** 도입하여 비동기 처리
- ML 모델 input을 위한 전처리가 매번 필요
  - 시계열 DB 도입 고려 + **SQL 쿼리에 전처리 과정 포함**

📚 **배운점**

- ML **실시간 추론**을 고려할 때에는 ML서버의 부담을 줄이고, 또 추론 속도를 개선해야 한다. 상황에 따라 알맞은 **DB 그리고 캐싱 전략**을 사용하도록 하자.