기계 상태 알림/확인 서비스

사이드 프로젝트

배경

2021년 군 출타보고/관리 서비스 창업을 마친 후 미래 커리어에 대한 고민을 하는 시기였습니다.

코로나로 인해 비대면 대학 수업 중에서는 엑티브한 활동에 제약을 느껴 새로운 자극이 필요했습니다.

이때 지인을 통해 공장 일자리를 소개 받았고, 우선 경제활동을 해보자는 생각으로 일을 시작했습니다.

당시 다뤘던 기계의 문제점을 IT 기술로 해결할 아이디어가 떠올라 이를 공장장님께 승인을 받고 시작하게 되었습니다.

시작 동기

24시간 내내 돌아가는 기계의 특성 상 퇴근 후 기계가 멈추면, 다음날 작업 일정이 밀리게 되는 손해가 발생했습니다.

이때 원격 제어를 통해서 다시 기계를 작동 시킬 수 있었지만, 퇴근 후에도 CCTV를 수시로 확인해야 했습니다.

이러한 부담을 덜고자 기계가 멈춘 시점을 앱 푸시 알림으로 알려주는 서비스를 개발했습니다.

타임 라인

버전	시작일	종료일	운영 공장/ 기계 수	Github
1.0	2022.01	2023.01	1개 / 2대	링크
1.1	2023.01	2023.09	2개 / 7대	
1.2	2023.07	2023.12	2개 / 7대	링크
2.0	2024.01	2024.02	2개 / 7대	링크

인사말

이제부터는 저와 서비스의 지난 2년 1개월간의 여정을 소개하려고 합니다.

기계 1대의 단 하루라도 낭비를 막을 수 있다면 의미가 있다고 생각하고 운영했습니다.

부디 흥미로운 시간이 되셨으면 좋겠습니다. 감사합니다.

버전 1.0 ~ 1.2 2022.01 ~ 2023.12

기능 구현에 집중

- 실시간 기계 상태 확인
- 기계 상태 변경 푸시 알림

기계 상태 데이터화

처음 고안한 방법 🌐 소스 코드 링크

작업중 화면 일부

작업 정지 화면 일부

1초 간격으로 무한 스크린샷, 해당 좌표 RGB값 비교하여 멈춤/작업 중 여부를 파이어베이스 DB에 전송

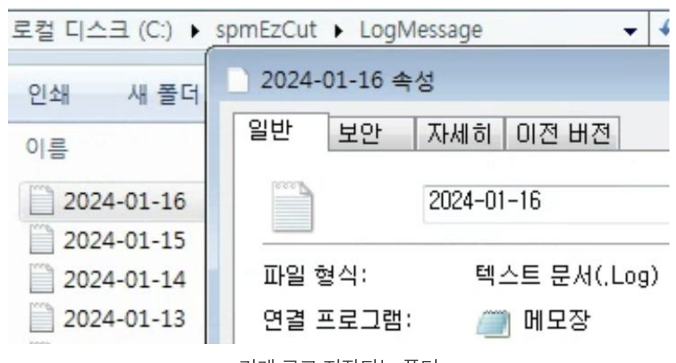
문제점

1. 새 창, 오류 창 등으로 무언가가 화면을 가리게 되었을 때 잘못된 데이터 전송 or 기계 상태 반영 불가
2. 멈춤/작업 중 여부 외에 구체적인 멈춤 사유를 알 수 없음

> 항상 정확한 기계 상태를 반영할 수 없는 통제 밖 변수들이 존재하는 구조

개선 🌐 소스 코드 링크

1. 작업자가 기계의 상태를 확인할 때 보는 기계 로그 자체를 가져올 수 있는지 확인
2. 기계가 실시간으로 '날짜.log' 파일에 로그를 저장하는 것을 발견
3. .log 파일을 실시간으로 트래킹하여 서버에 보내는 방식으로 개선



기계 로그 저장되는 폴더

기계 로그 메시지 내용

버전 1.0 ~ 1.2 2022.01 ~ 2023.12

개선효과

1. 제어할 수 없는 외부 변수와 분리하여 기계의 상태를 100% 반영
2. 기계 상태 외 작업 파일 이름, 가공 재료 두께 등 작업 상황에 대한 추가적인 정보 확인 가능

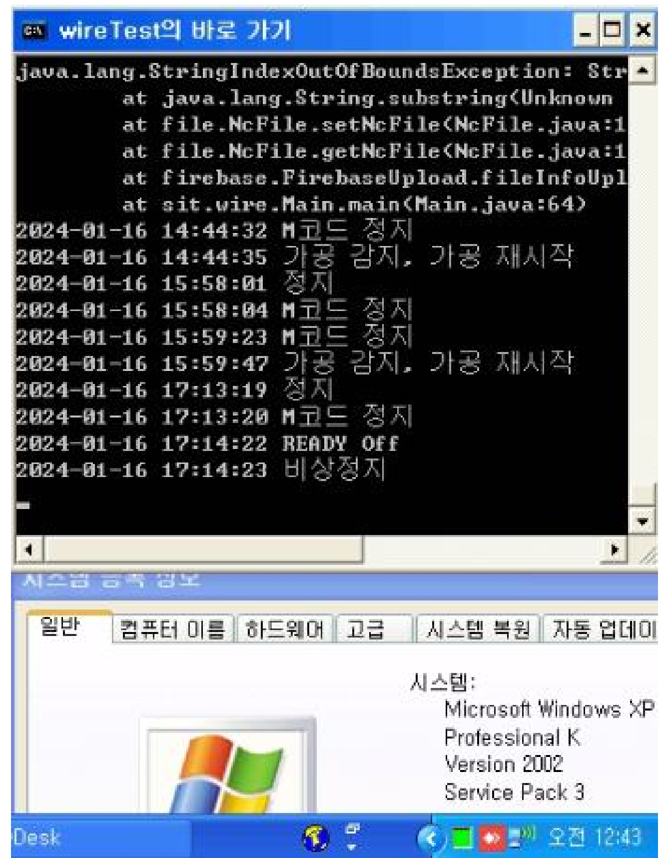
Windows XP 호환

새로운 챌린지

새로운 고객사 공장의 총 6대 기계 중 2대가 Windows XP인 상황

해결 과정 🌐 소스 코드 링크

1. Windows XP 에서 Firebase Firestore 에 데이터를 저장할 수 있는 방법 서치
2. XP 위의 Python, Java 버전에서는 Firestore 를 사용할 수 있는 Firebase SDK 미지원
3. 따라서 직접 Firestore REST API 를 호출하여 데이터를 저장하는 방법 선택
4. OkHttp3 를 활용하여 해당 Firestore DB URL 주소에 HTTP 요청을 보내는 로직 구현



Windows XP 사용 모습

App 화면

⇒ 이후 Java 8 에서 Firebase SDK를 지원함을 알게 되었지만 서비스 운영에는 문제가 없어 미변경

버전 2.0 2024.01 ~ 2024.02

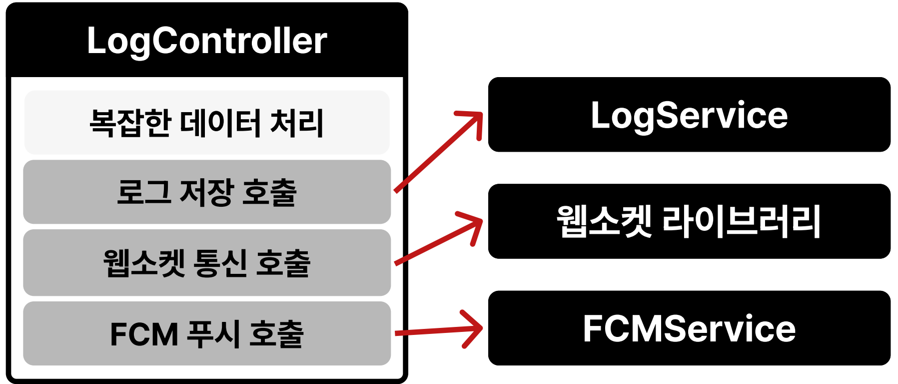
빠른 유지보수를 위한 프로젝트 구조 개선

- 버전 1.x : 기능 구현에 초점 ⇒ 시간이 지나면 알아보기 힘든 코드, 유지보수에 들어가는 시간적 비용 증가
- 최소한의 유지보수로 긴 기간 서비스를 제공하기 위해 수정이 쉽도록 코드를 개편
  - 패키지 구조와 API를 직관적으로 구성
  - 각 계층 / 기능 / 메서드의 역할을 분리
  - 이를 테스트 코드로 검증

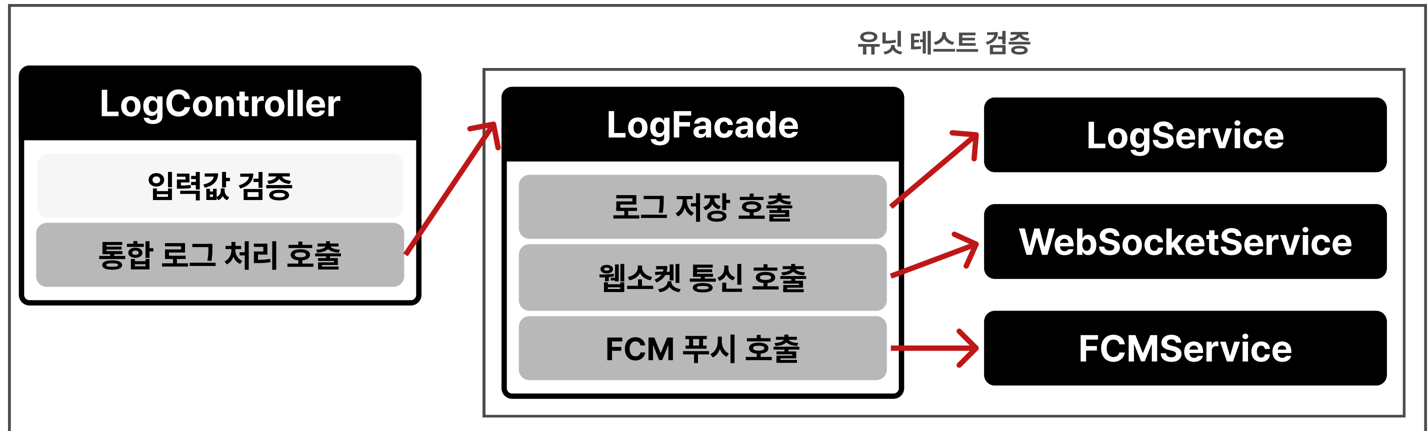
각 계층의 역할을 분리, 메서드를 검증하여 사용

구조 개선 소스 코드 링크

- **기존 구조**



- **새로운 구조**



- Business Layer 에서 비즈니스 로직과 그 실행 흐름을 담당하여 한 눈에 로직을 알아 볼 수 있도록 개선
- 각 비즈니스 로직을 유닛 테스트로 검증, 입력값/반환값을 명확히 정의하여 확장성 있는 코드로 개선

버전 2.0 2024.01 ~ 2024.02

비용 없이 서비스를 제공하기 위한 성능 개선

- 총 7대 기계 관리 중 DB 유료플랜을 쓰게 된 상황 발생
- 향후 더 발생할 고정 비용으로 인하여 서비스 유지에 차질 예상
- 이를 막기 위한 성능 개선 시도

과금 원인을 찾아 개선

클라우드 DB 무료플랜 초과 원인

1. 현재 DB 테이블 용량 약 9.4MB, JawsDB 무료 플랜 용량 5MB를 초과하여 INSERT 쿼리문 제한
2. 무료 플랜의 시간당 요청 쿼리문 한도 3,600개를 초과하여 DB 커넥션 제한

		Memory	Storage
$0/mo	Kitefin	Shared	5 MB
$10/mo	Leopard	Shared	1 GB

무료 플랜 Storage 용량

Name	Index_length	Data_length
dates	16384	16384
fcmtokens	49152	16384
logs	5816320	3686400
machines	16384	16384
members	32768	16384
processes	98304	114688

현재 DB 용량 = 약9.4MB

PLAN	MAX QUESTIONS
Single-tenant plans	No Limit
Kitefin	3,600/hr
Leopard	18,000/hr
Blacktip	36,000/hr

무료플랜 쿼리 한도

```
2024-01-18T12:04:04.333193+00:00 app[web.1]: Caused by: jav
a.sql.SQLSyntaxErrorException: User 'h2l6ld0hs40zgkvd' has
exceeded the 'max_questions' resource (current value: 3600)
```

쿼리 요청 횟수 초과

해결 방안

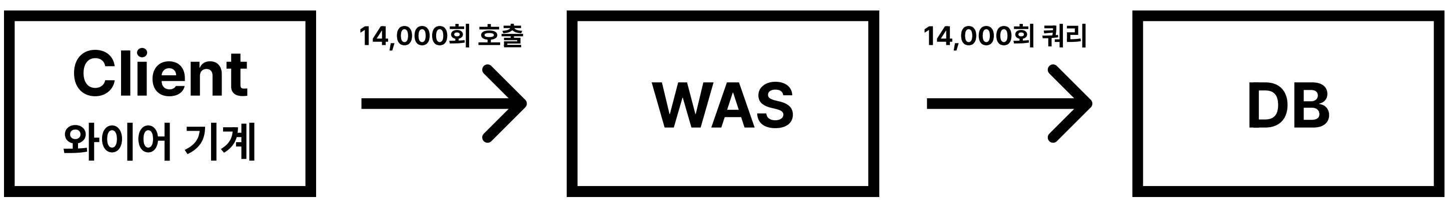
1. 충분한 용량을 제공하는 무료 클라우드 DB 서비스 사용 → 없음, 개선 불가
2. 핵심 비즈니스 로직인 기계 로그 저장 API 호출에 따른 쿼리 요청 횟수 최적화 → 2번 원인 개선
3. 쓰지 않는 핸드폰에 인프라를 직접 구축하여 클라우드 인프라 비용 제거 → 1, 2번 원인 개선

버전 2.0 2024.01 ~ 2024.02

핵심 비즈니스 로직 효율 개선

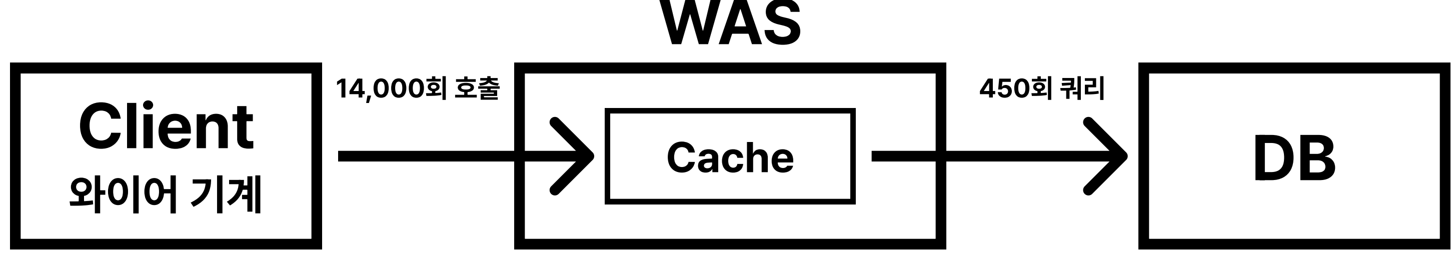
기존 로직 🌐 소스 코드 링크

- 기계당 월 평균 2,000회 기계 로그 저장 API 호출, 7대 기계에서 월 14,000회 API 호출
- 매 API 호출마다 DB INSERT 쿼리 발생



개선 🌐 소스 코드 링크

- 기계 로그들을 작업 단위로 묶어서 In-Memory 기반 ConcurrentHashMap 으로 Caching 구현
- 작업이 끝났을 때 Cache를 비우면서 한번에 DB로 INSERT



쿼리 수 14,000회 → 약 450회 / 97% 감소
기계 로그 저장 속도 53ms → 26ms / 50% 개선
기계 로그 조회 속도 37ms → 22ms / 41% 개선

Service	Count	Error	Total Elapsed(ms)	Avg Elapsed(ms)
/api/v1/logs/save<POST>	1,001	0	52,995	53

Service	Count	Error	Total Elapsed(ms)	Avg Elapsed(ms)
/api/v1/logs/save<POST>	1,000	0	26,417	26

추가로 고려한 점

- WAS 서버 재배포, 서버 kill 시 Cache 데이터 휘발의 위험
- 외부 캐시 저장소 사용 or Cache 데이터 백업 필요
- 과금 요소가 없도록 매 요청마다 기계 로그를 서버 내부 파일 시스템에 백업하여 해결

버전별 기술 현황

버전	1.0 / 1.1	1.2	2.0
개발/운영 기간	2022.01 ~ 2023.09	2023.07 ~ 2023.12	2024.01 ~ 2024.02
운영 공장 / 기계 수	1개 / 2대	2개 / 7대	2개 / 5대
Web Backend	Firebase Functions (serverless)	Java 11 Spring 5.3.27 Spring Boot 2.7.11 JDBC Template	**Java 17** **Spring 6.1.2** **Spring Boot 3.2.1** **JPA**
DB	Firebase FireStore	MySQL	MySQL
배포	Firebase	Heroku	On-premise
Web Frontend	-	Thymeleaf (SSR)	-
Windows App	Python / Java 8	C# Winform	Java 8 JavaFx
Mobile App	Flutter	Flutter	Flutter
Push Notification	FCM	FCM	FCM
기타	-	디자이너 참여	-

기술 사용, 마이그레이션 근거

VER 1.0

- 창업 씬에 몸 담으며 들어봤던 기술들 중 입문자 입장에서 가장 빠르게 구현할 수 있는 기술들을 선택

  ⇒ Python, Flutter, Firebase

VER 1.0 → VER 1.1

- 개인적인 주 언어를 Java로 선택하여 이에 익숙해지기 위해 Windows App을 Java 8로 마이그레이션

VER 1.1 → VER 1.2

- 전체적인 웹/모바일 애플리케이션 서비스 생태계를 이해하기 위한 기술들을 학습, 도입

  ⇒ Spring boot, Thymeleaf, MySQL, RestAPI
- 빠른 배포를 위해 러닝커브가 적절한 Heroku를 이용하여 배포
- 현업 Windows App 언어의 점유율을 근거로 C# Winform 선택

  ⇒ 현재 역량의 한계로 인한 유지보수 오버헤드가 누적중

VER 1.2 → VER 2.0

- 적은 유지보수로 긴 기간 서비스를 제공하기 위해 LTS 버전인

  Java 17 / Spring Boot 3.2 버전으로 마이그레이션
- 추후 개발 비용을 줄일 수 있고, 현재 현업에서 많이 쓰이는 ORM 기술인 JPA를 학습, 적용
- 인프라 비용을 줄이기 위해 직접 On-premise 환경 구축/운영
- C# Winform 으로 인하여 앞으로 발생할 오버헤드를 없애기 위해 익숙한 언어 기반인 JavaFx 로 마이그레이션