



MyHome Project



해당 포트폴리오는 필요성에 의한 프로젝트의 진행 과정으로 작성되었습니다.



프로젝트 소개

- 오래된 집에 IT를 접목하여 가족 구성원들이 조금 더 편리한 생활을 할 수 있도록 서비스를 제공하는 프로젝트입니다.



프로젝트 목표

- IoT를 접목하여 원격 제어 기능 제공
- 클라우드 서비스를 통해 사진, 영상 및 기타 파일 공유
- 모바일, 웹을 통하여 접근성 확대
- 지속적인 업데이트로 기능 추가

1 프로젝트 시작

IoT에 대해 라이브러리, 기술 등을 모르고 프로젝트를 도전하였기에 기술 검증과 기능 작동을 최우선으로 시작.

✓ 필요 사항 (에러 사항)

- 전등 원격 제어 서비스

📍 목표

- IoT 시스템 구축

🛠️ 해결 과정 및 오류

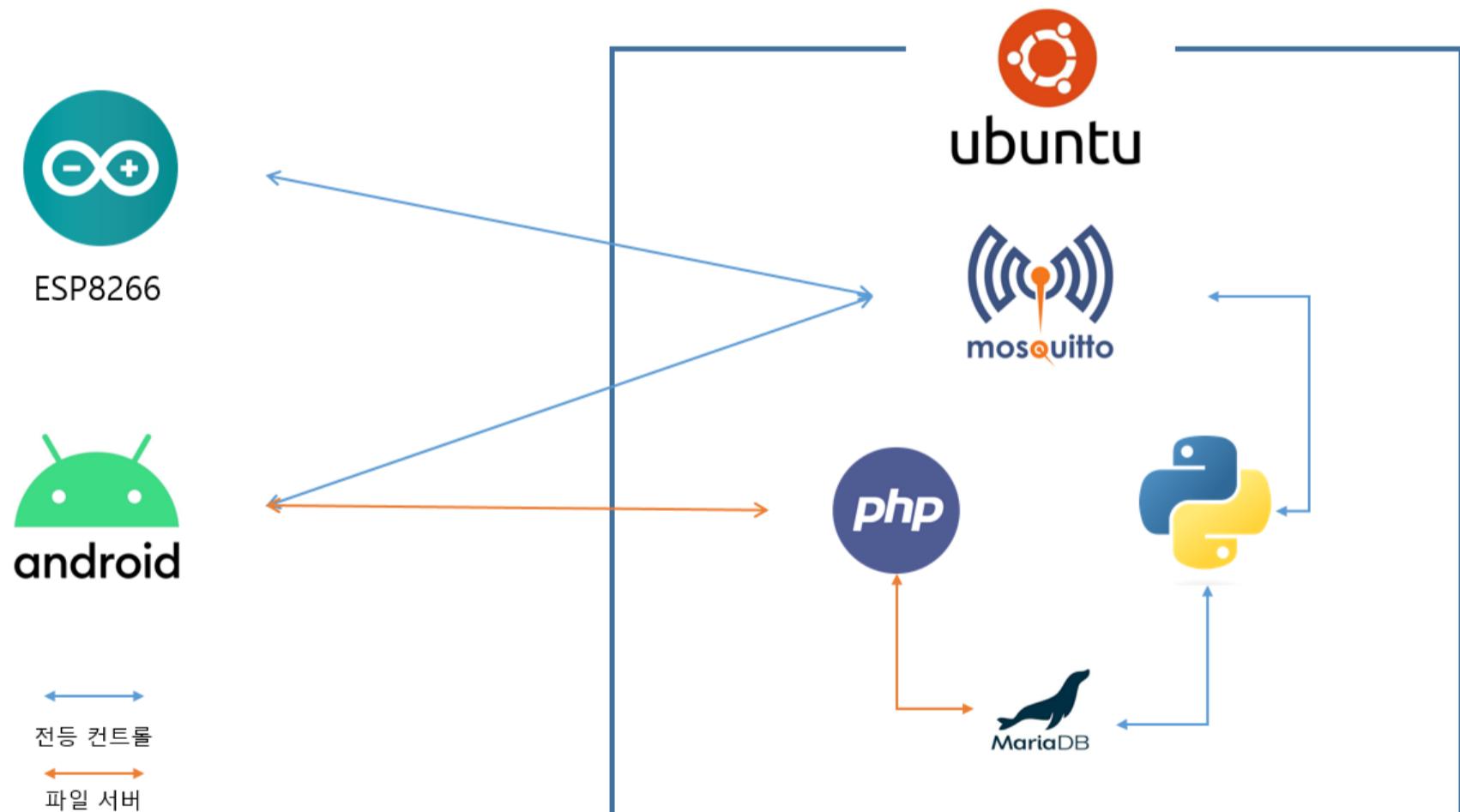
- 스위치(ESP8266)
 - 전등 ESP8266 칩셋 탑재된 저가형 스위치 구매 후 펌웨어 교체
 - MQTT는 서버와 연결, 버튼을 트리거로 하여 릴레이 작동 및 MQTT 발신 진행
- IoT 시스템 구축
 - 미니 PC 사용 Ubuntu 18.04 LTS 설치, 도메인 iptime 사용
 - Mosquitto MQTT 서버 구축
 - python으로 브로커 백엔드 구축
 - PHP로 안드로이드와 DB통신 백엔드 구축
- 안드로이드 어플리케이션
 - SDK 28, JAVA 네이티브 개발 (가족 구성원 모두 안드로이드 사용)

✓ 결과

- 안드로이드 폰에서 전등 원격 제어 성공
- 계정을 기반으로 한 작동 로그 정보 DB에 저장
- 로그 이미지

Day	Time	Room	Do	User
2021-9-15	2:49	small Room	Off	손준혁
2021-9-15	2:49	small Room	On	손준혁
2021-9-15	3:27	small Room	Off	손준혁
2021-9-15	6:44	bathRoom2	On	self
2021-9-15	6:50	bathRoom2	Off	self
2021-9-15	6:50	big Room1	On	self
2021-9-15	6:57	kitchen table	On	self
2021-9-15	7:2	kitchen table	Off	self
2021-9-15	7:27	bathRoom2	On	self

- 프로젝트 구조



🏁 깨달은 점 & 얻은 점

- 서버, 백엔드, 클라이언트를 전부 개발하면서 프로젝트의 구조를 학습함.
- 클라이언트와 DB간 데이터를 주고 받기 위해 백엔드가 어떤 역할을 해야하는지 학습함.
- 온라인 기반 서비스를 하기 위해서 DB에 보관해야 할 데이터, 클라이언트에 임시 보관할 데이터 구분하는 방법에 대해 경험함.
- MQTT를 통한 IoT 생태계 구축에 대한 경험을 쌓음.

2 부가 기능 추가

계획대로 흘러가지 않는 법. 새로운 기능 추가 요청을 만족하기 위해 워터풀 방식이 아닌, 그때그때 의견을 수렴하고 빠른 대처를 하는 방식으로 변경하여 도전

✓ 필요 사항 (에러 사항)

- 부가 기능 추가 요청

● 목표

- 전등 예약 작동 기능 추가
- 날씨 정보 기능 추가
- 공지사항 기능 추가
- 클라우드 기능 추가

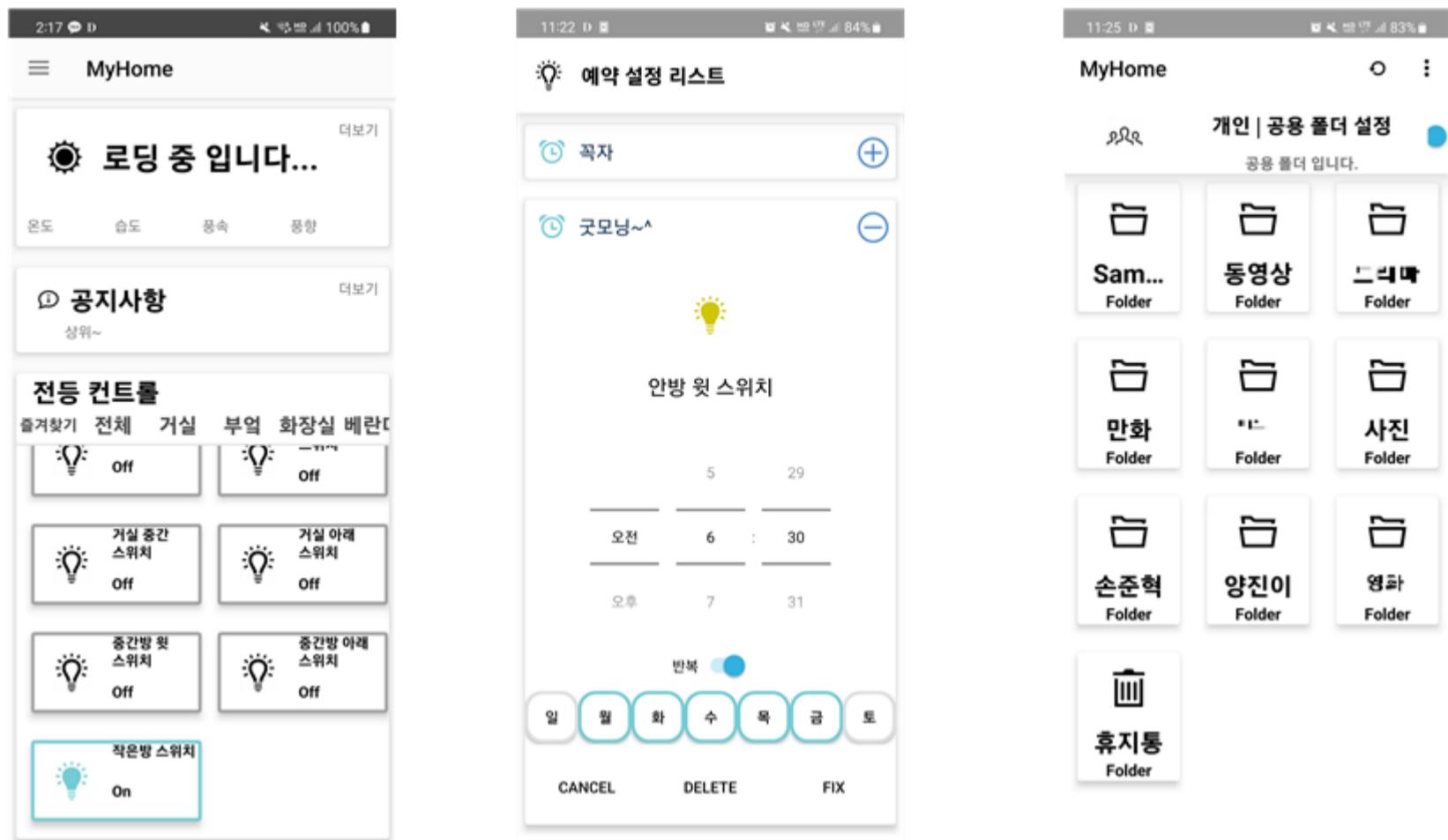
🛠️ 해결 과정 및 오류

- 전등 예약 작동 기능
 - Python schedule 라이브러리 사용
 - 실시간 스케줄 추가를 위해 MQTT를 트리거로 사용
 - 안드로이드에서 예약 DB 추가(PHP) 이후 MQTT로 메세지 발송
 - Python 백엔드에서 DB 데이터를 다시 탐색 및 기존 스케줄과 변경점 동기화
- 날씨 정보 기능
 - 기상청 API를 사용
 - 안드로이드에서 직접 호출하여 파싱하여 데이터 출력
 - API Url, 키 또한 코드로 내재
- 공지사항 기능
 - PHP를 이용한 DB 데이터 REST API 활용
- 클라우드 기능
 - SFTP를 통한 직접 제어
 - 안드로이드에서 명령어 전송으로 파일 이동, 삭제, 업로드, 다운로드 제어

📈 결과

- 예약 기능
 - 시간, 액션(on, off)의 예약 기능 추가
 - 요일, 반복 설정 추가로 옵션 제공
- 날씨 정보 기능
 - 현재 날씨를 메인에서 확인 가능
 - 날씨 세부 엑티비티에서 하루 날씨 확인 가능 및 지역 변경 가능
- 클라우드
 - 파일, 폴더 구분 및 서버에 존재하는 파일 실시간 정보 확인 가능.
 - 이동, 삭제, 업로드, 다운로드 가능
- 공지사항
 - 제목, 작성자, 내용, 시간 기반 공지사항 기능 제공
 - 메인에서 최신 공지 확인 가능

- 공지사항 세부 엑티비티에서 공지 리스트 확인 및 추가 공지 생성 가능
- 안드로이드 스크린샷



🏁 깨달은 점 & 얻은 점

- 처음 기획한 내용과 다른 내용이 추가될때 이를 수용하면서 개발하는 경험 쌓음.
- 스케줄을 통한 데이터 처리 과정 학습함.
- 데이터 처리를 해야하는 순서(ex: 로그인 이후 그에 해당하는 데이터 불러오기) 혹은 동기, 비동기 처리해야 할 부분을 판단하여 개발하는 경험 쌓음.

3 프로젝트 리팩토링

보안과 안정성 그리고 확장성까지 생각했을 때 한번 리팩토링을 할 필요가 있다고 판단. 언제 들어올지 모를 추가 요청을 대비하여 미래를 위한 투자 시작

✓ 필요 사항 (에러 사항)

- 보안 및 레거시 코드 리팩토링 필요

● 목표

- 보안을 위해 백엔드 전면 교체
- 날씨 정보 기능 업데이트 및 유지보수 위한 리팩토링
- 클라우드
- 안정성, 성능 향상, 확장성 개선을 위한 리팩토링
- 안드로이드 레거시 코드 업데이트
- 스위치의 서버 종속성 낮추기

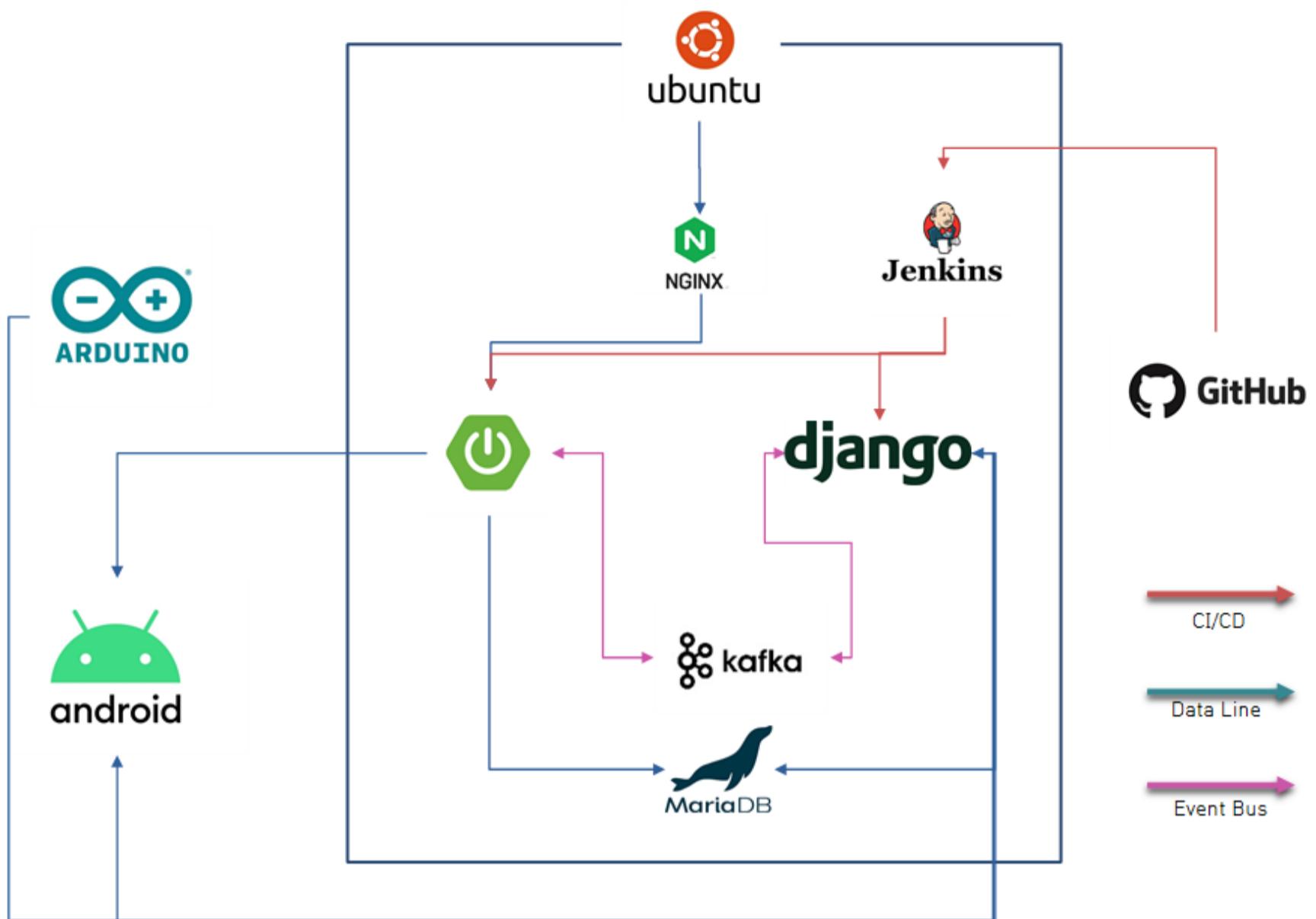
🛠 해결 과정 및 오류

- 보안 및 안정성 향상
 - 백엔드 프레임워크 도입
 - PHP → Spring Boot (JWT, Spring Security)
 - Python → Django
 - 권한 있는 계정을 통하여만 기능 호출 가능하도록 변경
- 날씨 제공 기능 리팩토링
 - API url 업데이트
 - 정보 처리 안드로이드 → Spring Boot
 - API 키 및 url DB로 이관
- 클라우드 기능 안정성, 성능 향상, 확장성 개선
 - SFTP 직접 제어 → Spring Boot, Django에서 처리.
 - 성능 향상
 - 백엔드에서 파일 정보(이름, 크기, 파일 타입, 위치 등)를 DB로 동기화
 - 클라이언트는 해당 정보를 표시
- 안정성
 - 클라이언트가 직접 서버 내부 접근 차단
 - RESTful API를 통해 권한 있는 요청만 처리
- 확장성 개선
 - 클라이언트에게 정보를 RESTful API로 제공함으로서 웹으로도 확장 가능하도록 함
- 안드로이드
 - SDK 28 → 32 업데이트
 - 레거시 코드 업데이트(AsyncTask → Retrofit2 등)
 - UI 업데이트

결과

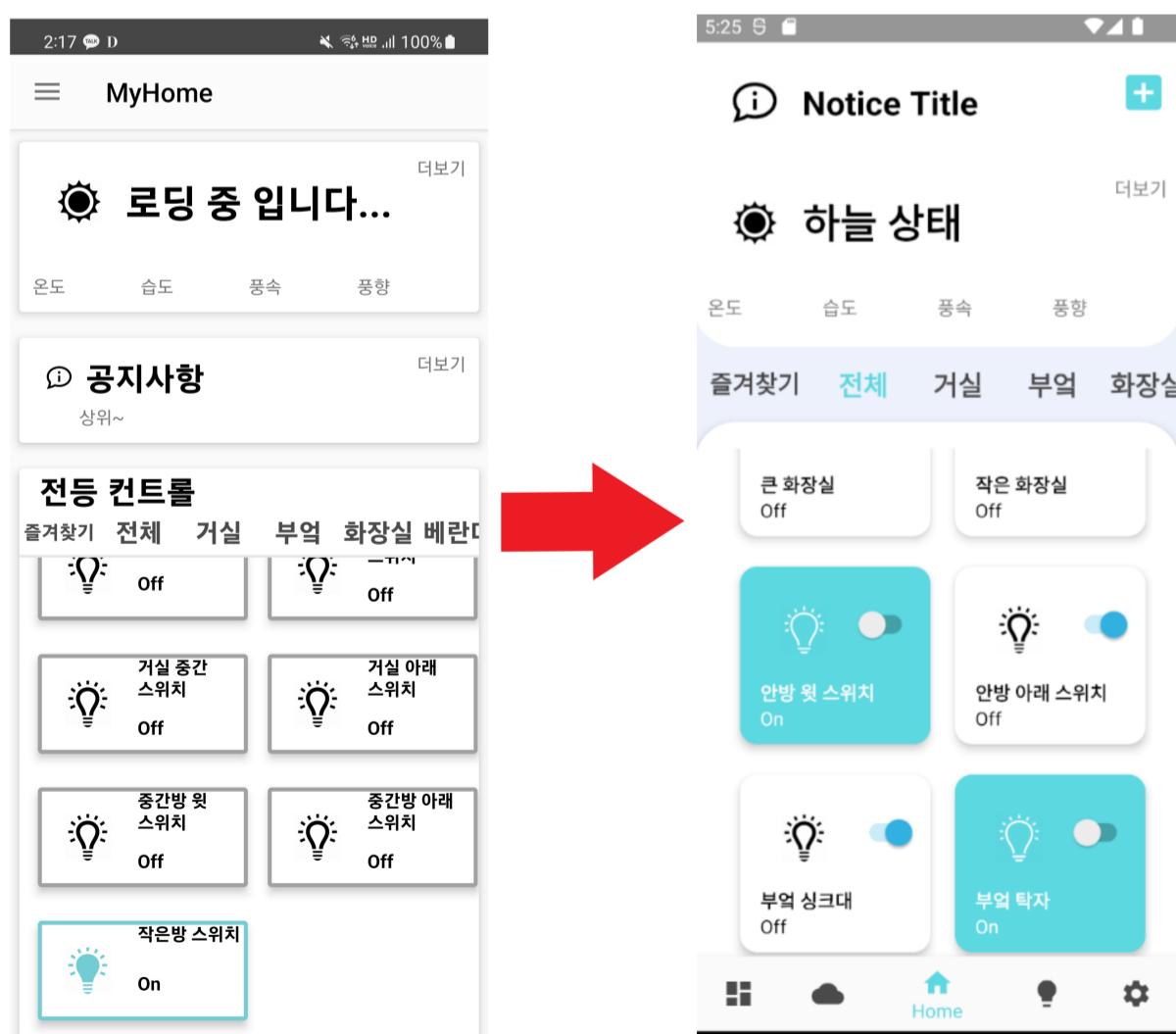
- 백엔드
 - 보안
 - 기존에는 요청할 경우 권한 상관없이 데이터 리턴되던 것을 계정 권한으로 구분하여 리턴
 - 계정 정보도 JWT를 통하여 인증 시 민감한 정보는 제외하고 인증이 가능하도록 함
 - 안정성
 - 파이썬 스크립트로 작동하는 백엔드는 예상치 못한 가벼운 에러로도 정지시 서비스 전체가 멈춰버리는 현상 발생, 이를 해결하기 위해 프레임워크를 활용하여 서비스 제공 안정성 향상
 - 날씨 기능 리팩토링
 - 기존에는 안드로이드에서 기상청 API에 직접 접근하여 데이터를 파싱 후 처리함으로서 서버에서도 처리 가능한 일을 모바일에서 불필요한 전력 소모 및 자원 소모가 존재함
 - API URL, 키 모두 내제되어 있어서 해당 값들이 변경 시 유지 보수가 까다로움
 - Spring Boot에서 데이터 로직을 처리하고 API URL, 키값을 DB로 이관하여 유지 보수 및 성능에서도 개선을 이뤄냄
- 클라우드 서비스
 - 보안
 - SFTP를 사용하여 직접 서버의 파일에 접근하는 제어권을 안드로이드에서 서버로 이관
 - 안드로이드에서 요청 시, RESTful API를 통해 서버에 요청하면 서버에서 처리하는 방식으로 변경
 - 파일 리스트 확인은 서버에서 DB에 파일 정보(이름, 타입, 크기, 위치 등)을 동기화하여 저장
 - 성능 향상
 - 파일 리스트 및 정보 탐색의 경우,
 - 기존에는 파일을 직접 확인하고 정보를 불러와 보여주는 방식이었으나, DB로부터 데이터를 받아와 보여주는 방식으로 변경 후 클라우드 저장소 역할하는 HDD의 부팅 및 로딩 시간만큼 단축하였음
 - 확장성 개선
 - 기존에는 안드로이드 ↔ 서버의 통신이 1:1로 맞춤 형식으로 구현된 기능들이 많았음(ex : SFTP)
 - 하지만 데이터 로직 부분을 서버로 이관하고 요청을 RESTful API로 대체하면서 모바일 뿐만 아니라 웹으로도 기능을 확장 할 수 있는 교두보를 만듬
- 안드로이드
 - SDK 업데이트
 - 28 → 32로 업데이트하면서 상위 OS에도 서비스 제공을 유지할 수 있음
 - 레거시 코드 업데이트
 - AsyncTask와 같이 deprecated된 코드들을 다른 코드 및 라이브러리로 변경
 - UI
 - 디자인이 기기 사이즈마다 다른것을 다시 디자인하여 유저의 기기에 맞추도록 개선
 - 메뉴를 타고 넘어가는 디자인에서 직관적으로 한 페이지에서 정보를 확인 할 수 있도록 개선
- 스위치
 - 기존에는 MQTT 서버와 연결이 끊기면 버튼도 작동을 하는데 딜레이가 길어 서버와 연결이 필수
 - 버튼 핸들러를 인터럽트로 처리하여 MQTT와의 연관성을 분리하여 MQTT 서버에 대한 종속성을 낮춤

- 프로젝트 구조

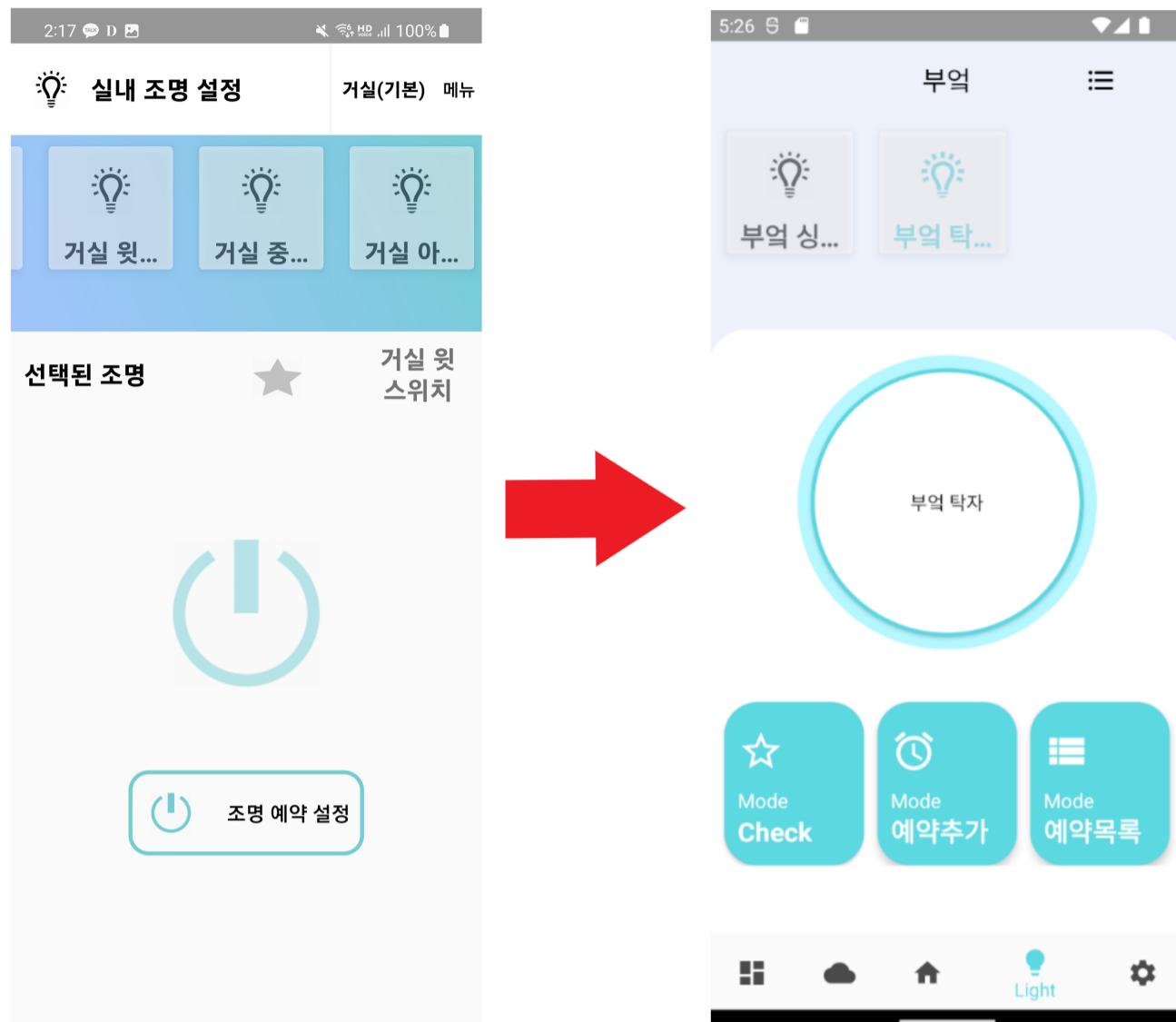


- 안드로이드 UI

- Main



- Light



🏁 깨달은 점 & 얻은 점

- 백엔드에서 REST API를 잘 정비해두면 클라이언트가 어떤 형태로든(모바일, 웹 등) 공통적인 부분을 많이 할 수 있어 서비스 확장, 유지보수에 큰 도움이 된다는 사실을 경험함
- 클라우드 뿐만 아니라 클라이언트가 서버에 있는 데이터, 파일을 직접 제어하는 것은 매우 위험하며 이를 우회하여 제어하도록 개발하는 경험을 얻음
- 백엔드를 개발 완료했다고 판단하여 모바일 개발을 넘어갔으나, 추가 기능이 필요한 부분이 생각보다 자주 일어남. 상황은 언제든 변경이 가능하며 이를 유연하게 받아들이고 대처하는 자세가 중요하다는 것을 경험함

4 서비스 장애 발생 대비 및 확장

집 근처 전기공사 덕에 정전이 자주 일어났다. 집에 있는 서버가 정전으로 인해 셧다운이 일어날 수 도 있다는건 생각했지만, 이렇게 자주 일어난다는 것은 고려조차 하지 않았다. 하지만 이를 대처해야 한다.

✓ 필요 사항 (에러 사항)

- 서비스 장애 안정화 및 접근성 강화

● 목표

- 집 근처 공사기간 잦은 정전으로 인한 서버 강제 재시작 횟수 증가.
- 이로 인한 백엔드 서비스의 잦은 장애를 극복하기 위한 자동 재실행 및 혹시 모를 서버 이관을 대비한 서비스를 도커 컨테이너화
- 부모님께서 큰 화면을 통하여 서비스를 사용하고 싶어하셔서 PC에서도 사용 가능하도록 웹을 추가하여 접근성 강화

🛠️ 해결 과정 및 오류

- 백엔드 안정화를 위한 컨테이너화
 - 서버 PC 자체가 고장이 날 수 도 있다라는 우려사항 때문에 서비스를 어느 리눅스 PC에서도 금방 복구가 가능하도록 추가 고려함
 - 앞선 이유로 리눅스 Service 등록과 도커 재시작 옵션 중 도커 재시작 옵션을 선택
 - 도커 실행 옵션 활용하여 서버 재시작 시, 서비스도 자동으로 재시작
 - 도커파일에서 환경을 설정하여 어디서든 일관된 환경을 제공
- 웹 추가로 서비스 접근성 강화
 - NextJS 기반으로 웹을 제작하여 모바일에서 제공하던 서비스를 그대로 제공

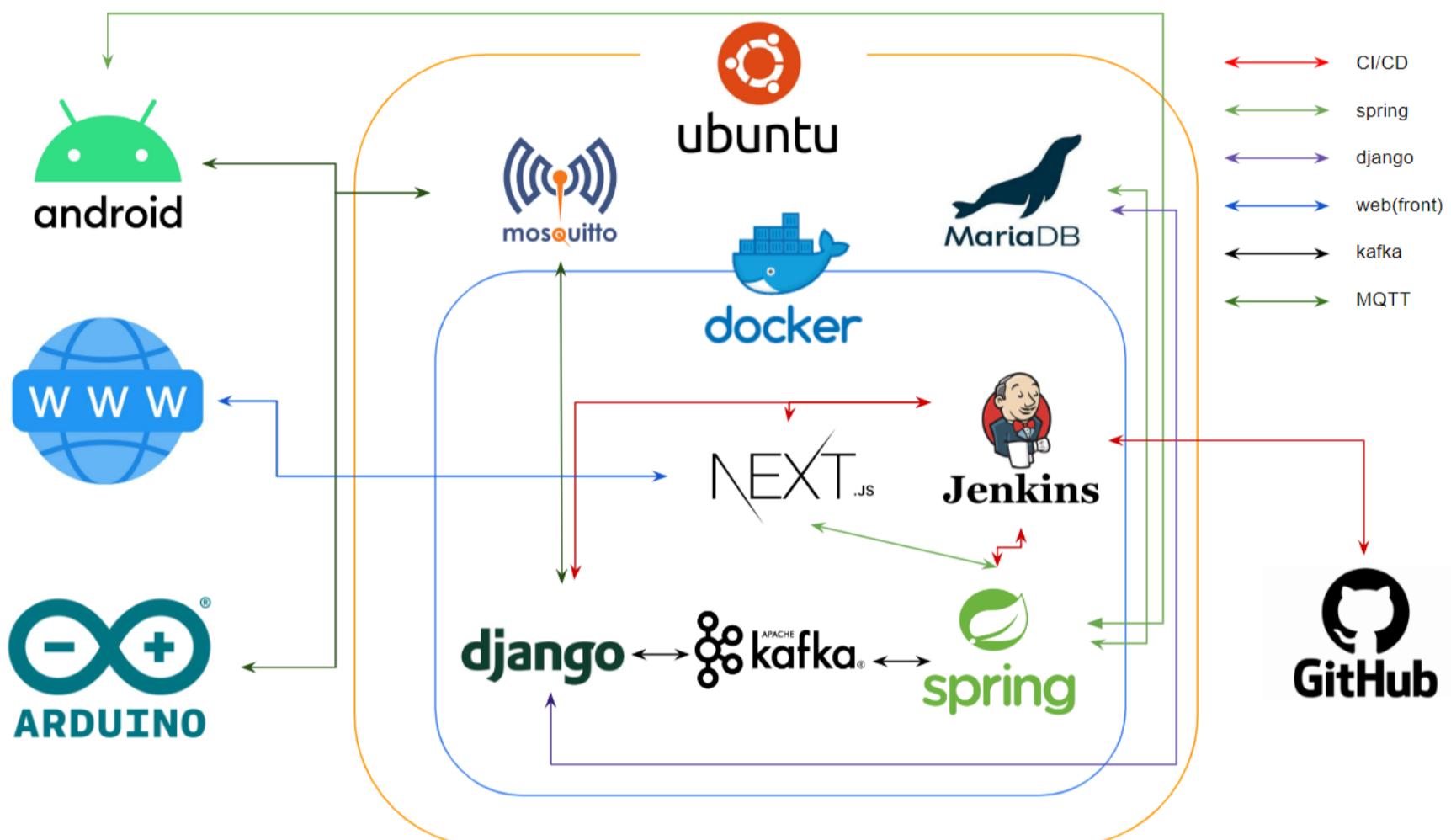
📈 결과

- 백엔드
 - 백엔드 코드에 도커파일을 통하여 도커 컨테이너로 배포
 - 도커 실행 옵션 중 —restart를 통하여 서버가 예상치 못하게 재부팅이 되어도 바로 재실행이 되도록 설정
 - 도커파일에서 환경을 설정하여(Java Or Python Version 등) 어디서든 일관된 환경을 제공
- 웹 서비스 추가
 - NextJS 13기반으로 웹 서비스 추가
 - 이전에 RESTful API로 서비스를 리팩토링 한 결과에 접목
- 이미지
 - 컨테이너로 작동되는 백엔드 서비스 리스트

```
sonjuhy@sonserver:~$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED             STATUS              PORTS     NAMES
4d508e1e6204   myhome_spring   "java -jar /app.jar"   20 hours ago    Up 20 hours        0.0.0.0:3001->3001
a02efdf29d846  wurstmeister/kafka  "start-kafka.sh"      2 weeks ago      Up 2 weeks         0.0.0.0:9092->9092
5434d4878108   wurstmeister/zookeeper  "/bin/sh -c '/usr/sb..."  2 weeks ago      Up 2 weeks         0.0.0.0:2181->2181
b96d7d2adb00   provectuslabs/kafka-ui  "/bin/sh -c 'java --..."  2 weeks ago      Up 2 weeks         0.0.0.0:9093->9093
1d93921cc57b   mariadb          "docker-entrypoint.s..."  6 weeks ago      Up 6 weeks        0.0.0.0:3306->3306
0b64d19c0bf3   jenkins/jenkins  "/usr/bin/tini -- /u..."  7 months ago     Up 7 months        0.0.0.0:8080->8080
```

STATUS	PORTS	NAMES
Up 20 hours	0.0.0.0:8081→8080/tcp, :::8081→8080/tcp	myhomeSpring
Up 2 weeks	0.0.0.0:9092→9092/tcp, :::9092→9092/tcp	kafka
Up 2 weeks	22/tcp, 2888/tcp, 3888/tcp, 0.0.0.0:2181→2181/tcp, :::2181→2181/tcp	zookeeper
Up 2 weeks	0.0.0.0:9093→8080/tcp, :::9093→8080/tcp	kafka-ui
Up 6 weeks	0.0.0.0:3306→3306/tcp, :::3306→3306/tcp	mariadb
Up 21 hours	50000/tcp, 0.0.0.0:9090→8080/tcp, :::9090→8080/tcp	jenkins

- 프로젝트 구조



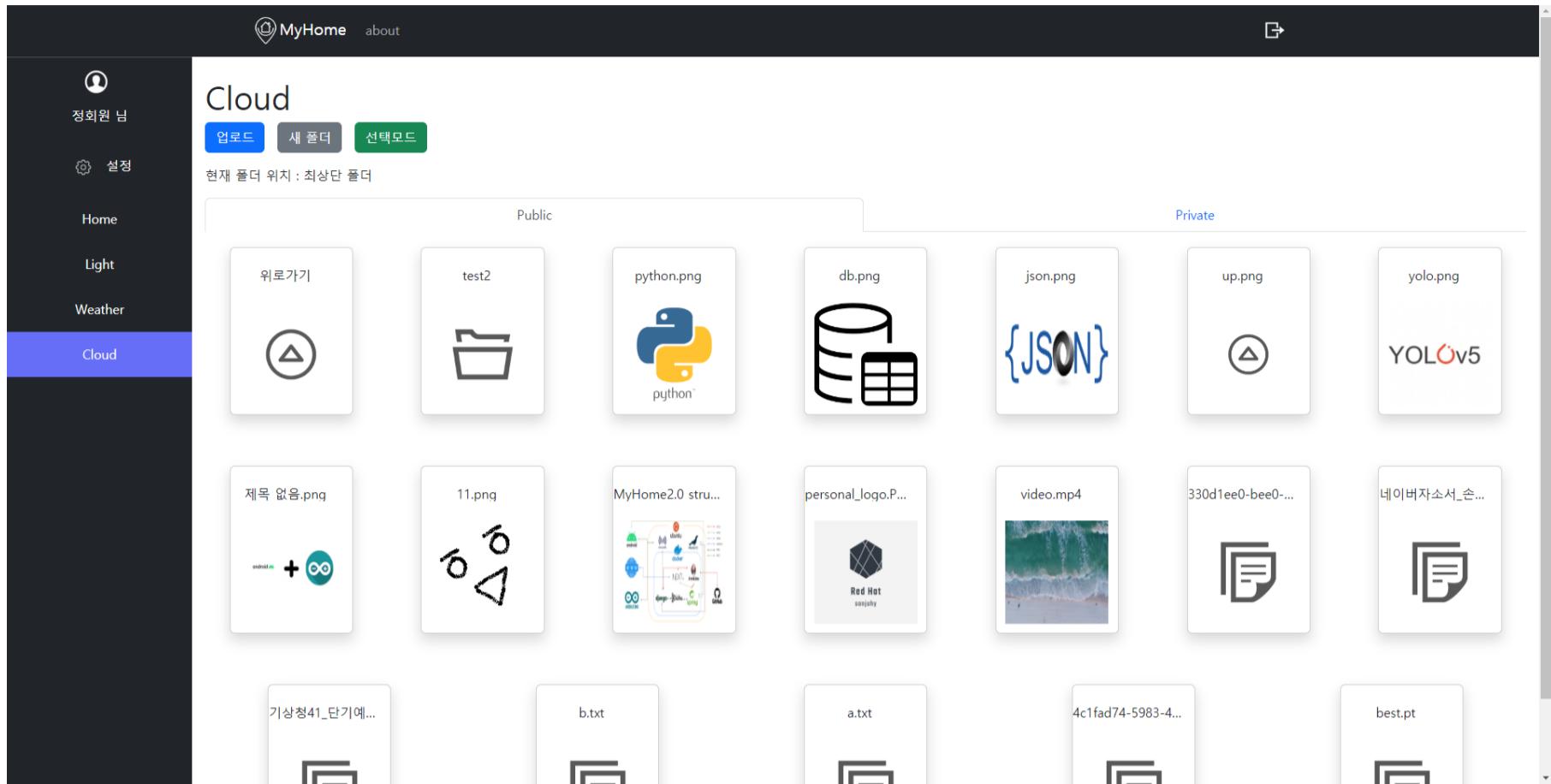
- 웹 사이트 스크린샷

- Main

The screenshot shows the main interface of the MyHome web application:

- Header:** MyHome, about
- Sidebar:**
 - 정회원 님 (Profile)
 - 설정 (Settings)
 - Home** (selected)
 - Light
 - Weather
 - Cloud
- Content Area:**
 - Notice:** Notice Title. This is a space that displays what you have written as a notice. — admin
 - Weather:** 맑음, 북풍 2.3m/s, 23 °C, 강수량 : 강수없음, 경상남도 창원시 성산구, 09:00 기준.
 - Light Control:** A row of seven light control buttons with labels: 작은 화장실, 안방 윗 스위치, 안방 아래 스위치, 부엌 싱크대, 부엌 탁자, 거실장, 거실 윗 스위치.

- Cloud



🏁 깨달은 점 & 얻은 점

- 웹과 모바일이 동일한 서비스를 제공할 때 어떻게 데이터를 공유하고 처리하는지 경험하였음
- 서비스에 예외사항이 생겨도 해당 서비스를 유지 할 수 있는 여러 방법을 찾아보고 적용하는 경험하였음
- 도커를 통해 서비스를 제공하는 법을 습득했으며, 컨테이너화 했을 때 가상화에 대해 유의할 점을 알고 적용하는 경험하였음.

5 로그 시스템 필요성 대두

새로운 환경, 더 많은 기능을 서비스하기에는 로그 없이는 한계를 느낍니다. 예러, 데이터 흐름을 파악하기 위해 로그는 이제 필수입니다.

✓ 필요 사항 (예러 사항)

- 서비스 중 발생하는 에러를 대응하기 위한 로그 시스템 필요성

📍 목표

- Kafka를 통해 발생한 로그를 수집하고 이를 확인 할 수 있는 플랫폼 구축

🛠️ 해결 과정 및 오류

- 수집하고자 하는 데이터를 선정하여 하나의 묶음(기본 단위)으로 설정
 - 백엔드 코드 중에서 데이터를 확인하고 싶은 체크포인트에 kafka를 통해 로그를 전송
- Kafka에 전송된 로그들을 수집, 정리 하는 서비스 구현
 - RabbitMQ 와 같이 다른 메세지 큐 프로그램도 고려하였으나, 로그 수집 서비스 조차 서버 다운 시 같이 다운이 될 수 있다는 점을 고려하여 메세지를 자체 저장 가능한 kafka로 선정
 - Spring Boot를 사용하여 토픽 별로 들어오는 로그를 MongoDB에 저장
 - 또한 로그 정보를 보여주는 웹이랑 통신할 REST API도 담당
 - 토픽 별 Document를 분리하여 데이터 저장
 - 초창기에는 하나의 Document에 데이터를 모두 저장하였음
 - 처음으로 로그를 DB로 옮기는 과정에서 비동기로 토픽에서 데이터를 받아 Write를 하려니 동시성 문제 때문에 분리하여 저장하는 것으로 변경
 - 로그를 볼 때 서비스 별로 보는 것이 우선 순위였기에 동시성 제어를 해결 후 하나의 Document에서 조건을 통해 가져오는 것보다 서비스 별로 들어오는 토픽을 기준으로 나누고 여기서 가져오는 것이 더 이득이라고 생각하여 여러 개로 구분하여 구현
- 수집된 로그 정보들을 확인할 웹 플랫폼 구현
 - NextJS 13 기반으로 구현하였으며, 오픈 템플릿을 사용하여 차트들을 통해 전체적인 데이터를 시각화 하여 확인 할 수 있게 하였음.

📈 결과

- 원하는 데이터
- 메세지 내 정보 : type, sender, service, content
- Kafka 메세지 자체 정보 : 메세지 발행 시간

Key	Value	Headers	Timestamp	2023. 11. 21. 0시 0분 0초
	<pre>{"id": 0, "type": true, "sender": "Spring", "service": "Cloud-Check", "content": "[traversalFolder(private)] file (dto) : FileServerPublicDto(path = _home_disk1_home_public_E_driver, name=E_driver, uuidName=61ea646c-a18b-361c-8131 -7a2db7610a7b, type=dir, size=8.0, location=_home_disk1_home_public_, state=1, deleteStatus=0), no exist file"}</pre>		Timestamp type: CREATE_TIME	

- Kafka에 전송된 로그들

	Offset	Partition	Timestamp	Key	Preview	Value	Preview
+	11506	0	2023. 11. 21. 0시 0분 0초			{"id":0,"type":true,"sender":"Spring","service":"Clo...	
+	11507	0	2023. 11. 21. 0시 0분 0초			{"id":0,"type":true,"sender":"Spring","service":"Clo...	
+	11508	0	2023. 11. 21. 0시 0분 0초			{"id":0,"type":true,"sender":"Spring","service":"Clo...	
+	11509	0	2023. 11. 21. 0시 0분 0초			{"id":0,"type":true,"sender":"Spring","service":"Clo...	
+	11510	0	2023. 11. 21. 0시 0분 0초			{"id":0,"type":true,"sender":"Spring","service":"Clo...	
+	11511	0	2023. 11. 21. 0시 0분 0초			{"id":0,"type":true,"sender":"Spring","service":"Clo...	
+	11512	0	2023. 11. 21. 0시 0분 0초			{"id":0,"type":true,"sender":"Spring","service":"Clo...	
+	11513	0	2023. 11. 21. 0시 0분 0초			{"id":0,"type":true,"sender":"Spring","service":"Clo...	
+	11514	0	2023. 11. 21. 0시 0분 0초			{"id":0,"type":true,"sender":"Spring","service":"Clo...	
+	11515	0	2023. 11. 21. 0시 0분 0초			{"id":0,"type":true,"sender":"Spring","service":"Clo...	
+	11516	0	2023. 11. 21. 0시 0분 0초			{"id":0,"type":true,"sender":"Spring","service":"Clo...	
+	11517	0	2023. 11. 21. 0시 0분 0초			{"id":0,"type":true,"sender":"Spring","service":"Clo...	
+	11518	0	2023. 11. 21. 0시 0분 0초			{"id":0,"type":true,"sender":"Spring","service":"Clo...	
+	11519	0	2023. 11. 21. 0시 0분 0초			{"id":0,"type":true,"sender":"Spring","service":"Clo...	
+	11520	0	2023. 11. 21. 0시 0분 0초			{"id":0,"type":true,"sender":"Spring","service":"Clo...	

- 로그를 수집하여 MongoDB에 저장
- 토픽 별 Document를 분리하여 데이터 저장

cloud_check_log

Storage size: 2.47 MB **Documents:** 23 K **Avg. document size:** 552.00 B

cloud_log

Storage size: 20.48 kB **Documents:** 14 **Avg. document size:** 335.00 B

database_sequence

Storage size: 20.48 kB **Documents:** 6 **Avg. document size:** 42.00 B

default_log

Storage size: 1.02 MB **Documents:** 10 K **Avg. document size:** 527.00 B

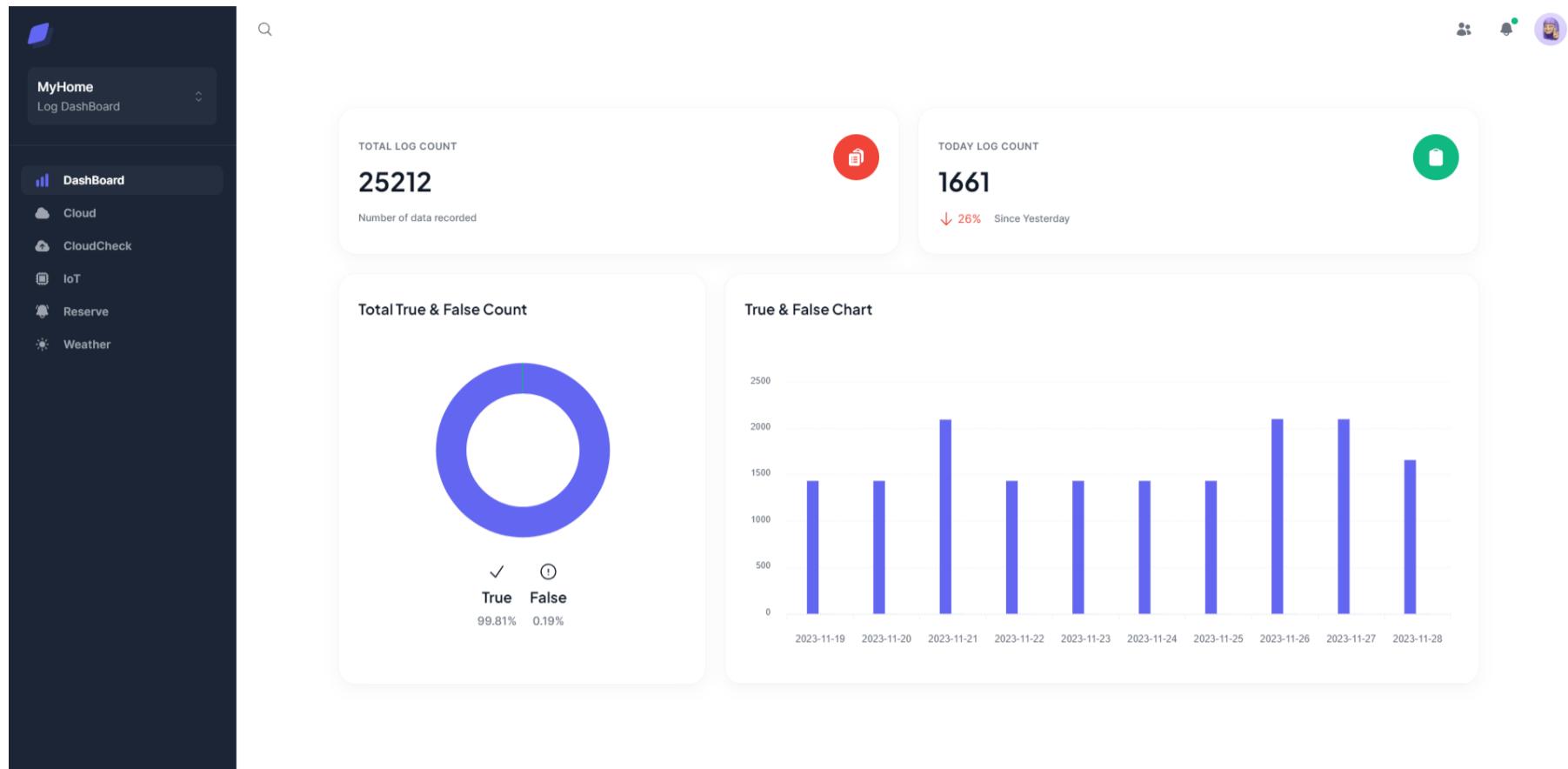
iot_log

Storage size: 20.48 kB **Documents:** 14 **Avg. document size:** 255.00 B

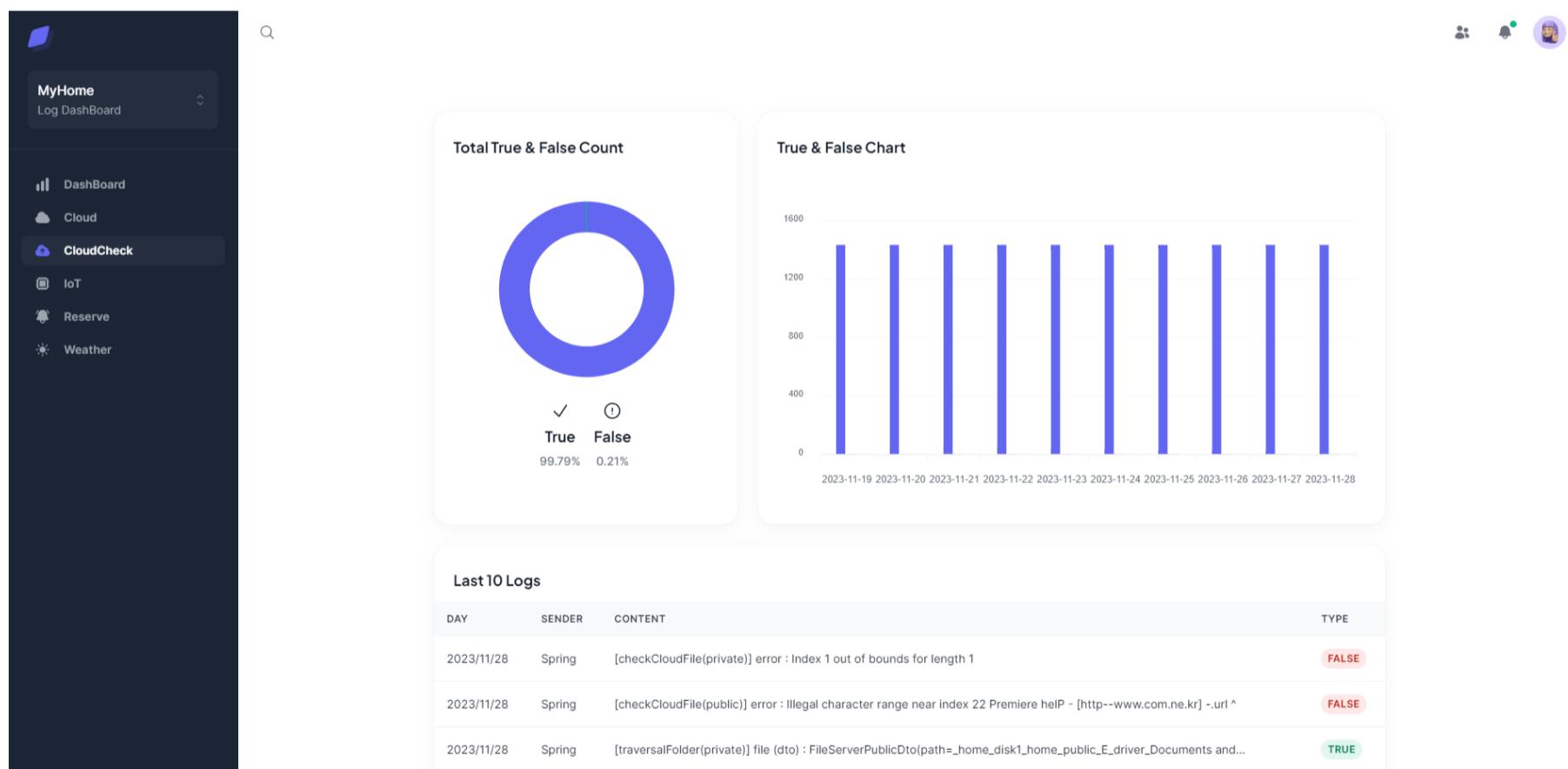
reserve_log

Storage size: 20.48 kB **Documents:** 2 **Avg. document size:** 233.00 B

- 웹 플랫폼
- 서비스별 상세 페이지를 통해 대략적인 데이터를 확인 가능하며, 디테일 페이지를 통해 로그들의 세부 정보 뿐만 아니라 조건 검색 기능을 추가하여 효과적으로 로그들을 확인 할 수 있도록 함.
- Main

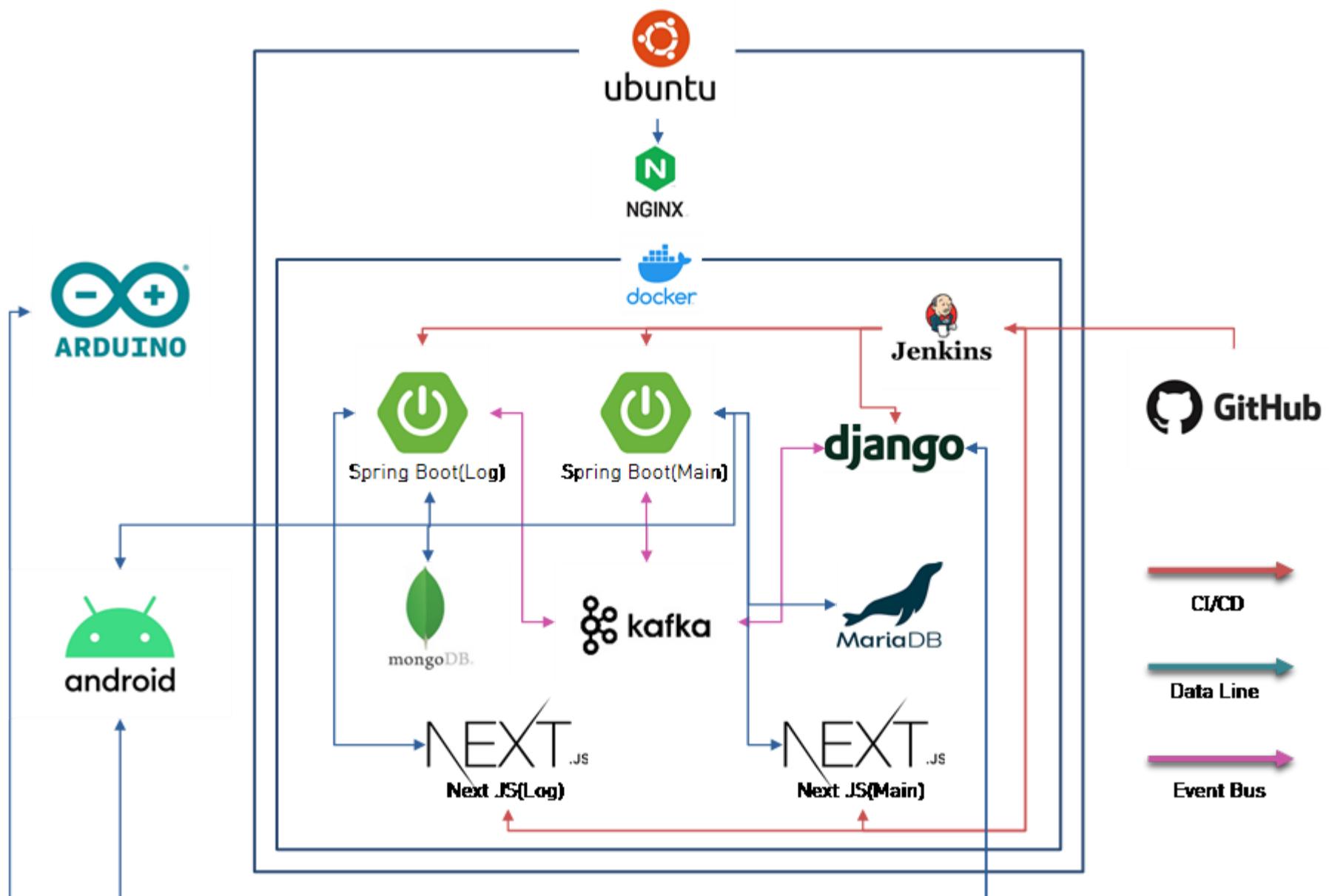


- Service Main



- Service Detail log

- 구조도



🏁 깨달은 점 & 얻은 점

- 로그 수집 및 분석할 수 있는 서비스가 왜 필요한지 몸소 깨닫고 구현함
- 로그를 수집할 때 기준과 타입을 정하는 것이 어렵고 중요하다는 것을 경험함
- 어떤 목적을 가지고 로그를 수집할 건지에 따라 수집 형태, 데이터 유형 등 전부 달라지며 이를 검토하여 서비스를 제공해야 한다는 것을 경험함

6 대용량 데이터 처리

클라우드 서비스에 5천개 미만이였던 파일이 갑작스레 300만개 이상으로 증가했다. 기존 방식으로는 DB에 데이터를 넣는 데 오래 걸린다. 원활한 서비스 제공을 위해 줄여야 한다.

✓ 필요사항

- 갑작스레 늘어난 파일들(5천개 미만 → 300만개 이상)을 탐색하고 해당 데이터를 DB에 넣는 시간이 너무 오래 걸림.

● 목표

- 약 300만개의 파일들을 빠르게 탐색 완료
- 파일 데이터들을 DB에 insert를 빠르게 완료하는 것
- 종합 과정을 5분 이내로 끝낼 것

🛠 해결 과정 및 오류

기존 방식

- 파일 탐색 방식
 - DFS 방식으로 탐색 폴더일 경우 깊이를 내려감.
- DB 입력 방식
 - 파일 하나당 INSERT 실행
 - JPA를 통해 saveAll()를 사용

변경된 방식

- 파일 탐색 방식
 - Files.walk() 방식으로 스트림으로 데이터를 받고 이를 DTO로 변경하여 리스트 타입으로 정렬
- DB 입력 방식
 - Bulk Insert 형식으로 입력 진행
 - JDBCTemplate의 batchUpdate() 사용

📈 결과

- 테스트 환경
 - CPU : i7-8750H
 - RAM : 16GB
 - DB : MySQL 8.0
 - Framework : Spring Boot(3.1.3)
- 탐색시간

	변경 전	변경 후	성능 개선 수치
5천개	2초	0초(소수점 추정)	2초
14만개	21초	2초	950%
300만개	299초	50초	498%

- INSERT 시간

	변경 전	변경 후	성능 개선 수치
5천개	5초	0초 (92ms)	5,334%
14만개	1분 16초	2초	3,700%
300만개	17분 18초	58초	1,689%

- 데이터 300만개 기준 시간 단축 정도

- 탐색시간 : 1/6 수준으로 단축
 - DB INSERT 시간 : 1/19 수준으로 단축

🏁 깨달은 점 & 배운점

- Files.walk()를 이용하여 파일 정보를 불러오는 것이 여러번 I/O 호출하는 것보다 많이 빠르다는 점.
- DB에 데이터를 대량으로 넣을 경우 I/O 호출을 최소화 하는것이 성능 향상에 엄청 큰 영향을 미침.
 - save() : DB 호출 횟수, Spring에서 @Transaction 생성 횟수 모두 n회 반복하여 n이 클수록 불리
 - saveAll() : DB 호출 횟수 n회 반복, Spring에서 @Transaction 생성은 한번만 하여 save()보다 유리
 - batchUpdate() : (전체 데이터 수 / 쿼리에 담을 수 있는 만큼 담은 수)만큼 DB 호출
 - ex) 전체 데이터 320만개 기준 44회 호출