

# 도소현

Contact : 010-6732-2315

E-mail : lyny.doh@gmail.com

Blog : <https://velog.io/@devlyny>

## 서비스의 가치를 생각하는 개발자

서비스의 안정적인 운영에 대해 함께 고민하는 것을 좋아합니다.

사용자에게 매일 닿을 수 있게 서비스의 가치를 만드는 백엔드 개발자를 꿈꾸고 있습니다.

활동에 대한 증빙자료는 이 링크를 통해 확인할 수 있습니다.



# Education

---

이화여자대학교

GPA (3.78/4.3)

호크마교양대학 (2020.03 - 2021.03)

컴퓨터공학과 졸업예정(2021.03 - 2024.02)

**EPITA(School of Engineering and Computer Science)**

Summer Program AI Course (2022.07)

## 전공 관련 수료 과목

2020 - 2  
1학년 2학기

- 컴퓨터 프로그래밍 및 실습

2021 - 1  
2학년 1학기

- 객체지향 프로그래밍 및 실습
- 오토마타 및 형식언어
- 사이버보안개론
- 소프트웨어창의융합설계

2021 - 2  
2학년 2학기

- 컴퓨터구조
- 임베디드 시스템 및 실험
- 시스템 SW 및 실습
- JAVA 프로그래밍 및 실습

2022 - 1  
3학년 1학기

2022 - 2  
3학년 2학기

- 데이터베이스
- 정보통신공학
- 운영체제

2023 - 1  
4학년 1학기

- 기계학습
- 블록체인응용
- 자료구조
- 컴퓨터네트워크

2023 - 2  
4학년 2학기

- 인공지능
- 컴파일러

- 클라우드컴퓨팅
- 빅데이터응용
- 오픈SW플랫폼(FastAPI & Vanilla JS 프로젝트 진행)

# Projects

## MILE <https://www.milewriting.com>

글 모임을 위한 글쓰기 공간, 마일



## Stack

- Java 17, SpringBoot 3.2.1 JPA, MySQL, Redis AWS Console, Docker, QuertDsl, Github Actions, Swagger,

## GitHub

<https://github.com/Mile-Writings>

## 역할

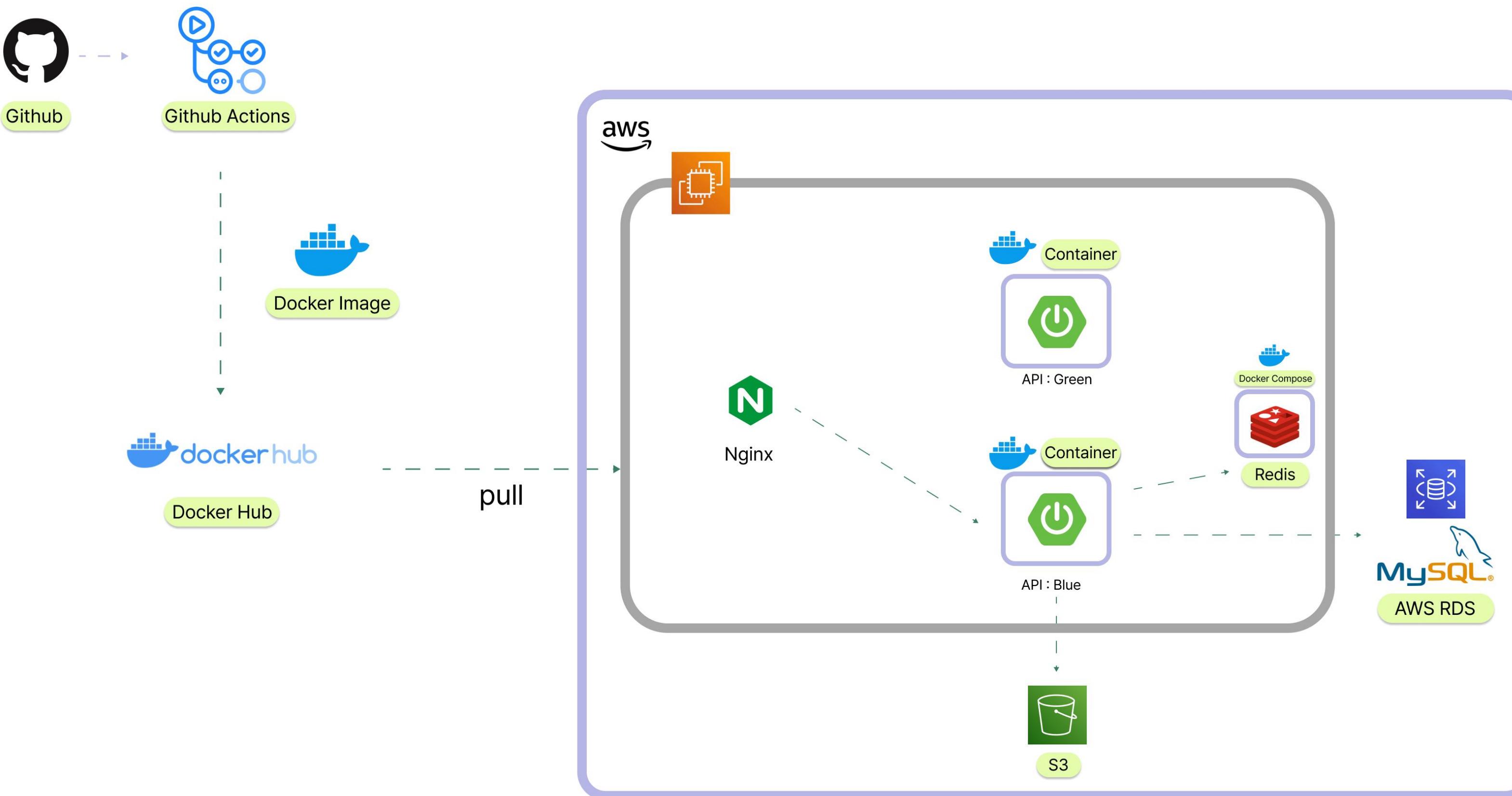
- 리드 서버 개발자
- Docker 컨테이너 기반 배포 환경 구축
- Openfeign을 이용한 소셜 로그인, Security + JWT + Redis 기반의 인증/인가 프로세스 구축
- Spring Boot 기반 API 작성
- Nginx를 이용해 리버스 프록시 구축
- 멀티 모듈 프로젝트를 통해 모듈간 의존성 분리
- MDCUtil & Discord 연동을 통한 에러 및 요청/응답 객체 로깅
- Redisson 라이브러리의 setnx 활용을 통해 중복 체크 관련 API 분산 락 구현

# Projects

**MILE** <https://www.milewriting.com>

서버 아키텍처

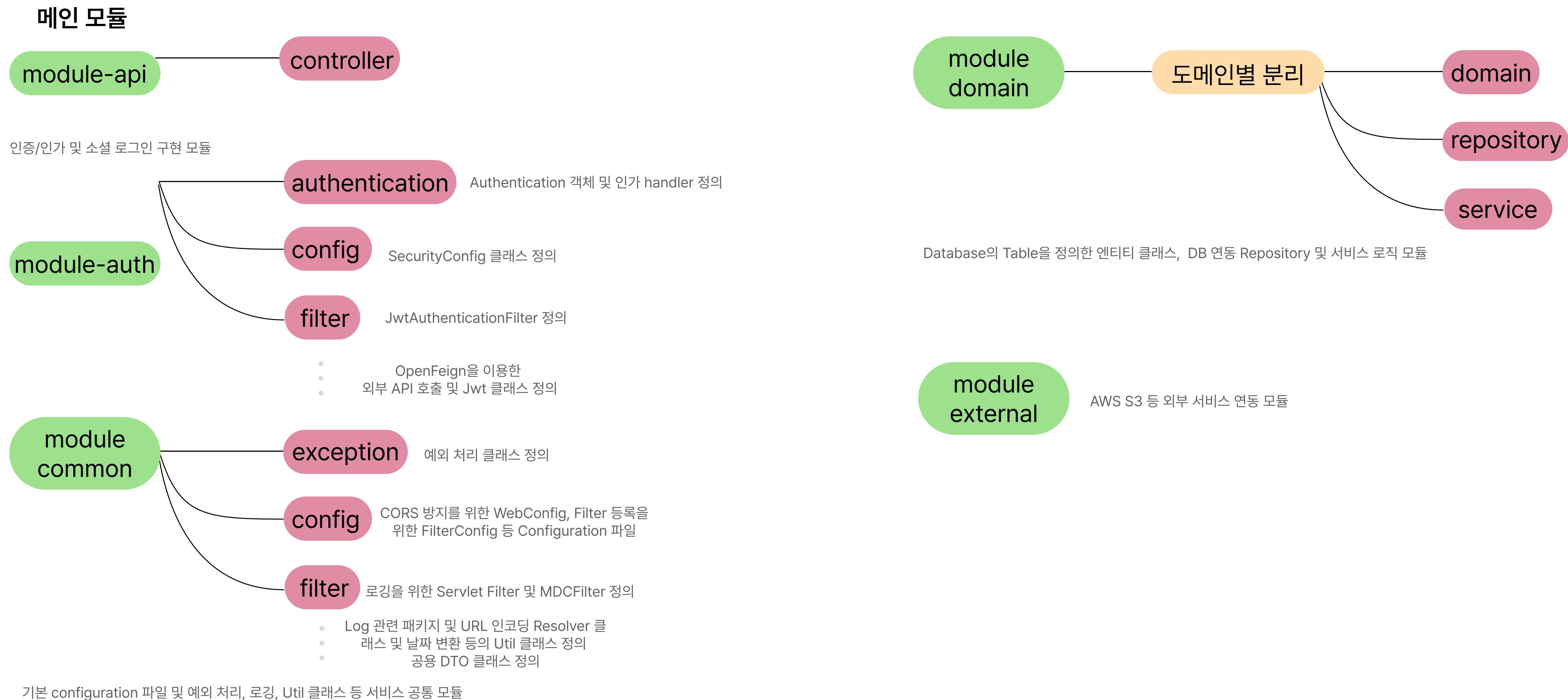
 Mile Architecture Diagram



# Projects

## MILE <https://www.milewriting.com>

### 프로젝트 구조



# Projects

## MiLE <https://www.milewriting.com>

### 구현 상세

멀티 모듈 프로젝트를 통해 모듈간 의존성 분리

### 멀티 모듈 도입 배경

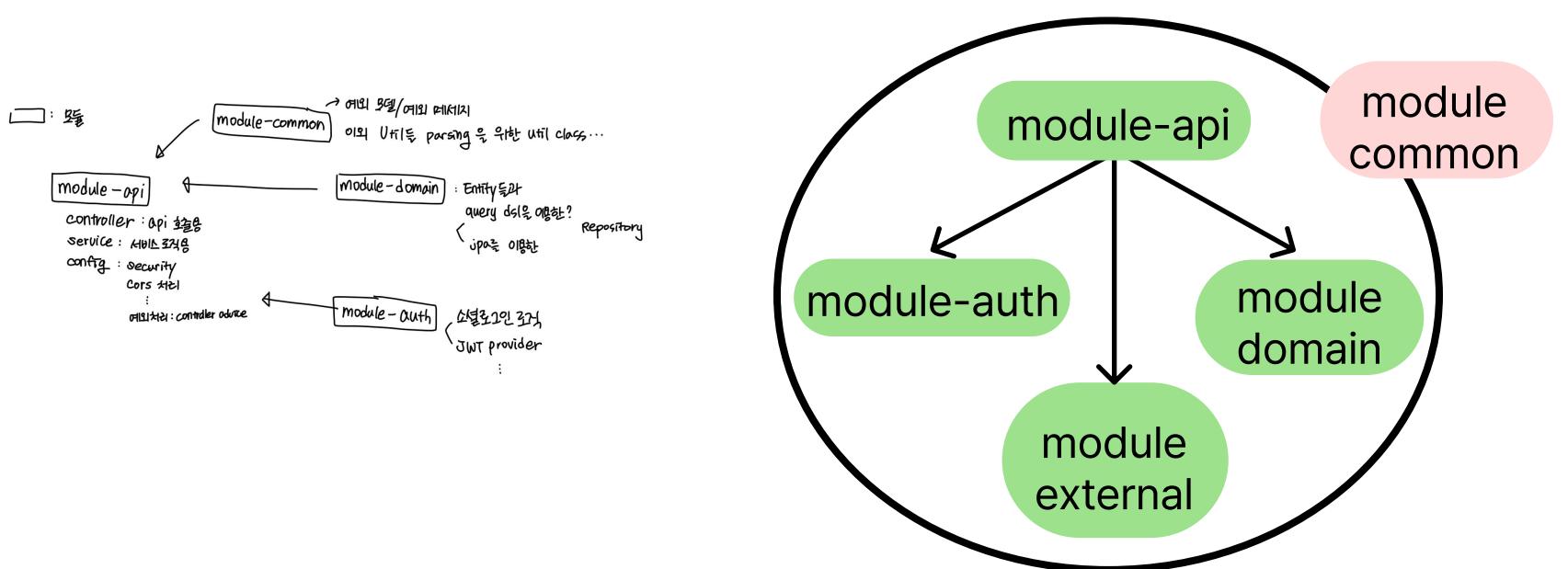
패키지간 의존성을 줄이는 방법에 대해 고민하던 중, MiLE에서는 패키지가 가지는 역할을 모듈화하여 관리하고자 했습니다.

### 멀티 모듈 도입 방식

팀원에게 멀티 모듈 방식에 대해 설명한 후 함께 구조도를 그렸습니다.

### 초기 구성

### 현재 구성



### 도입 결과

코드 수정이 있을 때 한 모듈의 수정 이후 사이드 이펙트가 100% 줄었습니다.

### 구현 상세

delete 쿼리 개선

### 도입 배경

프로젝트의 리팩토링 과정에서 글 삭제의 경우 글에 해당하는 좋아요(궁금해요)와 댓글 삭제 쿼리가 delete ? where id = ?의 형식으로 무한대까지 나가는 상황인 것을 발견했습니다.

### 도입 방법

JPQL을 이용하여 delete from 쿼리를 이용해 벌크 연산을 실행했습니다.

JPQL:  
@Query("delete from Comment c where c.post = :post")  
void deleteAllByPost(@Param("post")final Post post);

### 도입 결과

리팩토링 이전  
최대 무한대의 쿼리문 발생 가능

리팩토링 이후  
최대 3개의 delete 문으로 감소

# Projects

## MILE <https://www.milewriting.com>

### 구현 상세

#### MDCFilter를 이용한 Discord 로깅

##### 도입 배경

Logback을 사용해 Discord에 로깅을 하고 있던 상황에서 여러 번의 요청이 동시에 들어올 때 멀티 스레드 환경에서 로그가 섞여 요청에 대한 분석이 어려웠습니다.

##### 도입 결과

##### 기존 Discord 로깅 형태

```
1e3hFOYR-9MRGNTe-phemrPmT53hbA, access-control-allow-origin=, sec-ch-ua='Not_A
Brand';v="8", "Chromium";v="120", x-amzn-trace-id=Root=1-65a7c20d-27393f46259037614ec57df0,
sec-ch-ua-mobile=0, sec-ch-ua-platform="macOS", host=api.milewriting.com, connection=close,
accept-encoding=gzip, deflate, br, user-agent=Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36, sec-fetch-dest=empty}
2024-01-17 21:03:25.848 [32m[http-nio-8080-exec-10] [0;39m [1;31mERROR [0;39m
[1;37m[com.mile.aspect.LoggingAspect.requestErrorLevelLogging: [33m45 [0;39m] [0;39m - ===>
Response: <500 INTERNAL_SERVER_ERROR Internal Server Error,ErrorResponse[status=500,
message=서버 내부 오류입니다.],[]>
2024-01-17 21:03:25.854 [32m[http-nio-8080-exec-10] [0;39m [1;31mERROR [0;39m
[1;37m[com.mile.aspect.LoggingAspect.requestErrorLevelLogging: [33m47 [0;39m] [0;39m -
=====END=====
=====

2024-01-17 21:03:25.846 [32m[http-nio-8080-exec-3] [0;39m [1;31mERROR [0;39m
[1;37m[com.mile.aspect.LoggingAspect.requestErrorLevelLogging: [33m45 [0;39m] [0;39m - ===>
Response: <500 INTERNAL_SERVER_ERROR Internal Server Error,ErrorResponse[status=500,
message=서버 내부 오류입니다.],[]>
2024-01-17 21:03:25.857 [32m[http-nio-8080-exec-3] [0;39m [1;31mERROR [0;39m
[1;37m[com.mile.aspect.LoggingAspect.requestErrorLevelLogging: [33m47 [0;39m] [0;39m -
=====END=====
=====

2024-01-17 21:03:29.251 [32m[http-nio-8080-exec-1] [0;39m [1;31mERROR [0;39m
```

##### 문제점

로깅의 시작점과 끝점이 섞여서 로그가 어떤 요청에 해당하는 것인지 알 수 없었습니다.

##### 도입 방식

요청 별로 저장한 MDC에 대하여 Discord Appender를 통해 Discord Webhook Uri로 정돈된 형태로 로깅을 진행하였습니다.

MDCFilter를 통해 HTTP 요청으로부터 필요한 정보를 가져와 JSON 형식으로 변환하여 MDC에 저장 한 후 , HTTP 요청과 관련된 컨텍스트 정보를 로그에 추가할 수 있도록 만들었습니다.

##### 도입 후 Discord 로깅 형태

여러개의 요청에 대한 에러가 발생해도 MDC 별로 정돈된 형태로 Discord Webhook을 통해 로깅됩니다.

```
2:43 [ERROR - 문제 간략 내용]
java.lang.IllegalArgumentException: Last unit does not have enough valid bits
[Exception Level]
ERROR
[문제 발생 시간]
2024-01-19 02:43:40
[이용자 요청 URI 정보]
"/api/post/undefined/authenticate"
[이용자 IP 정보]
"1.225.181.100, 172.31.33.9"
[HTTP 헤더 정보]
"sec-fetch-mode": "cors",
"referer": "http://localhost:5173",
"sec-fetch-site": "cross-site",
"x-forwarded-proto": "https",
"accept-language": "ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7",
"origin": "http://localhost:5173",
"x-forwarded-port": "443",
"x-forwarded-for": "1.225.181.100, 172.31.33.9",
"accept": "application/json, text/plain, */",
"x-real-ip": "172.31.33.9",
"access-control-allow-origin": "*",
"authorization": "Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9eyJ0YXQiOiE3MDU1NTgyNTYsImV4cCI6MTcwNjct2Nzg1NiwiWtYmVySWQiOjxfQ_UPM2vMdyJpBPMQHCLnGDN5DwH
```

```
[Exception 상세 내용]
java.lang.IllegalArgumentException: Last unit does not have enough valid bits
at java.base/java.util.Base64$Decoder.decode0(Base64.java:868)
at java.base/java.util.Base64$Decoder.decode(Base64.java:566)
at java.base/java.util.Base64$Decoder.decode(Base64.java:589)
at com.mile.utils.SecureUrlUtil.decodeUrl(SecureUrlUtil.java:18)
at com.mile.resolver.post.PostVariableResolver.resolveArgument(PostVariableResolver.java:36)
at org.springframework.web.method.support.HandlerMethodArgumentResolverComposite.resolveArgument(HandlerMethodArgumentResolverComposite.java:122)
at org.springframework.web.method.support.InvocableHandlerMethod.getMethodArgumentValues(InvocableHandlerMethod.java:226)
at org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:179)
at org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:118)
at org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.invokeHandlerMethod(RequestMappingHandlerAdapter.java:917)
```

# Projects

---

## MILE <https://www.milewriting.com>

### 구현 상세

글모임 이름 중복 체크 관련 분산락 적용

### 구현 동기

베타 테스트 유치를 앞두고 PM과 함께 테스트 대상 유저를 200명으로 예상했습니다. FGI(Focus Group Interview) 결과, 글모임 이름 중복 체크 과정에 대한 데이터 정합성 처리가 필요하다는 점을 확인했습니다.

### 구현 방법

1. 글모임 생성 및 글모임 이름 중복체크에 활용되는 메서드에 적용할 어노테이션(@AtomicValidateUniqueMoimName)을 생성
2. 트랜잭션 커밋 후 락 해제 시점을 보장하기 위해 Propagation.REQUIRES\_NEW 옵션을 지정해 부모 트랜잭션의 유무에 관계없이 별도의 트랜잭션으로 동작하게끔 설정
3. Around 어노테이션을 통해 @AtomicValidateUniqueMoimName이 적용되어 있는 메서드에 대해 Redisson setnx를 사용하여 분산 락을 획득하고 해제하는 로직 작성

### 테스트

ExecutorService를 활용하여 100개의 동시 요청이 들어왔을 때 초반에 들어온 1개의 요청을 제외한 나머지 요청은 중복체크에 걸려 400 Bad Request의 상태 코드를 리턴하는 상황을 가정, MockMvc로 작성

# Projects

---

## MILE <https://www.milewriting.com>

### 구현 상세

전략 패턴을 사용한 소셜 로그인 구현

### 구현 동기

여러 개의 서비스를 제공하여 로그인 안정성을 제공하고자 새로운 서비스를 추가하였습니다. 새로운 엔드포인트를 추가하는 방향일 경우 소셜 로그인 서비스가 추가될 경우 확장성이 다소 낮다고 고려했습니다.

### 구현 방식 관련 Pull Request

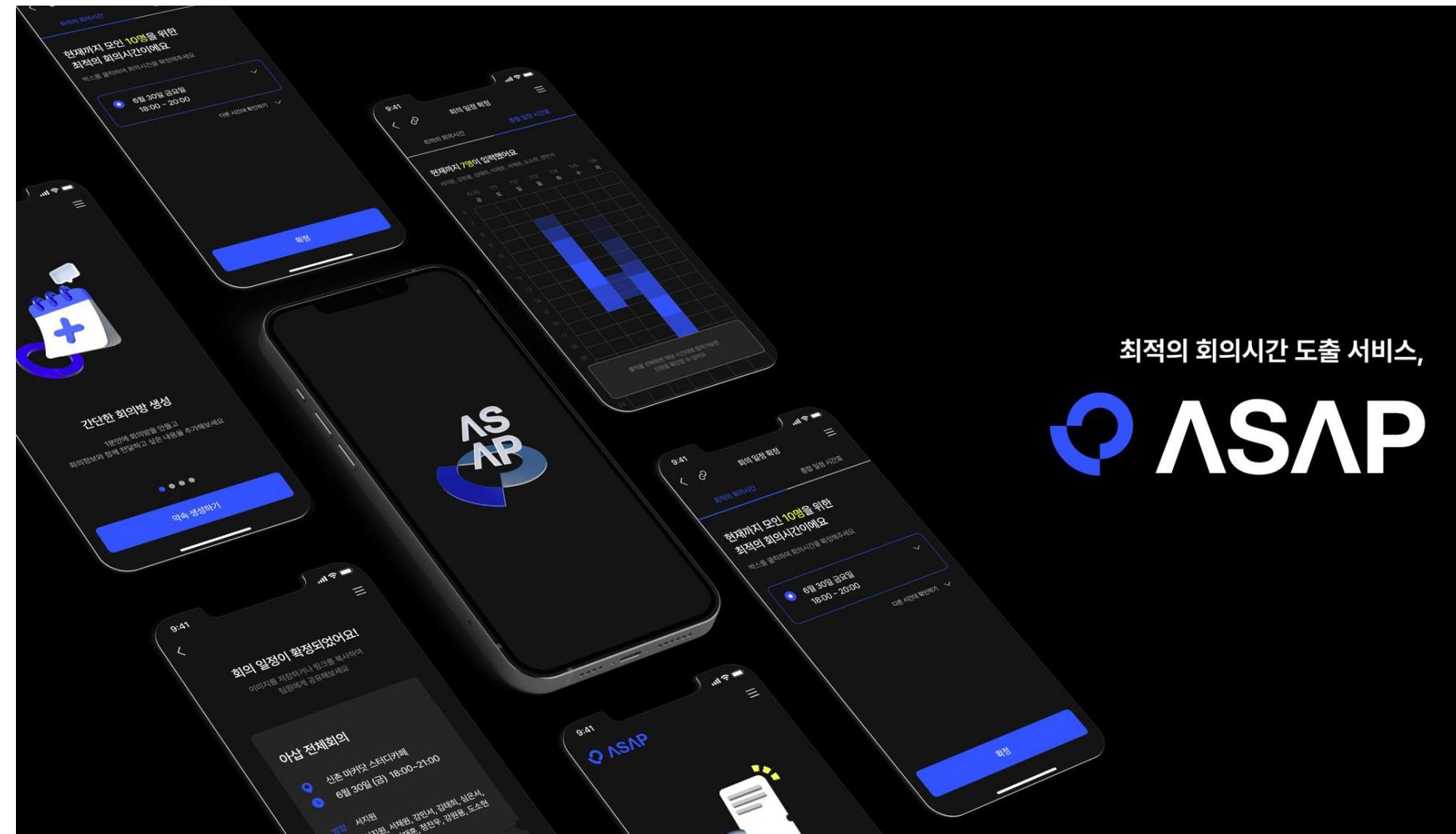
1. 런타임 시 알고리즘을 바꾸는 전략패턴을 적용
2. 로그인 전략의 툴을 Interface로 분리하여 이를 구현한 전략 클래스(Kakao, Google)를 구현, 각 전략 클래스에 구분을 위해 SocialType을 Getter와 함께 정의
3. 등록된 빈 클래스를 런타임에 따라 매칭해주는 매니저를 구현하여 매니저 클래스에 다음과 같이 Map으로 의존성을 받고 생성자를 통해 의존성을 주입받고, 각 전략 클래스마다 정의되어 있는 Social Type을 키로 보유
4. 실제 런타임에서는 switch, case 문으로 아래와 같이 LoginStrategy를 대응

# Projects

---

## ASAP <https://www.beginwithasap.com>

최적의 회의시간 도출 서비스,  
IT 창업동아리 SOPT APPJAM 우수상 수상



## Stack

- Java 17, SpringBoot 3.2.1 JPA, MySQL, AWS Console, Github Actions, Swagger

## GitHub

<https://github.com/ASAP-as-soon-as-possible>

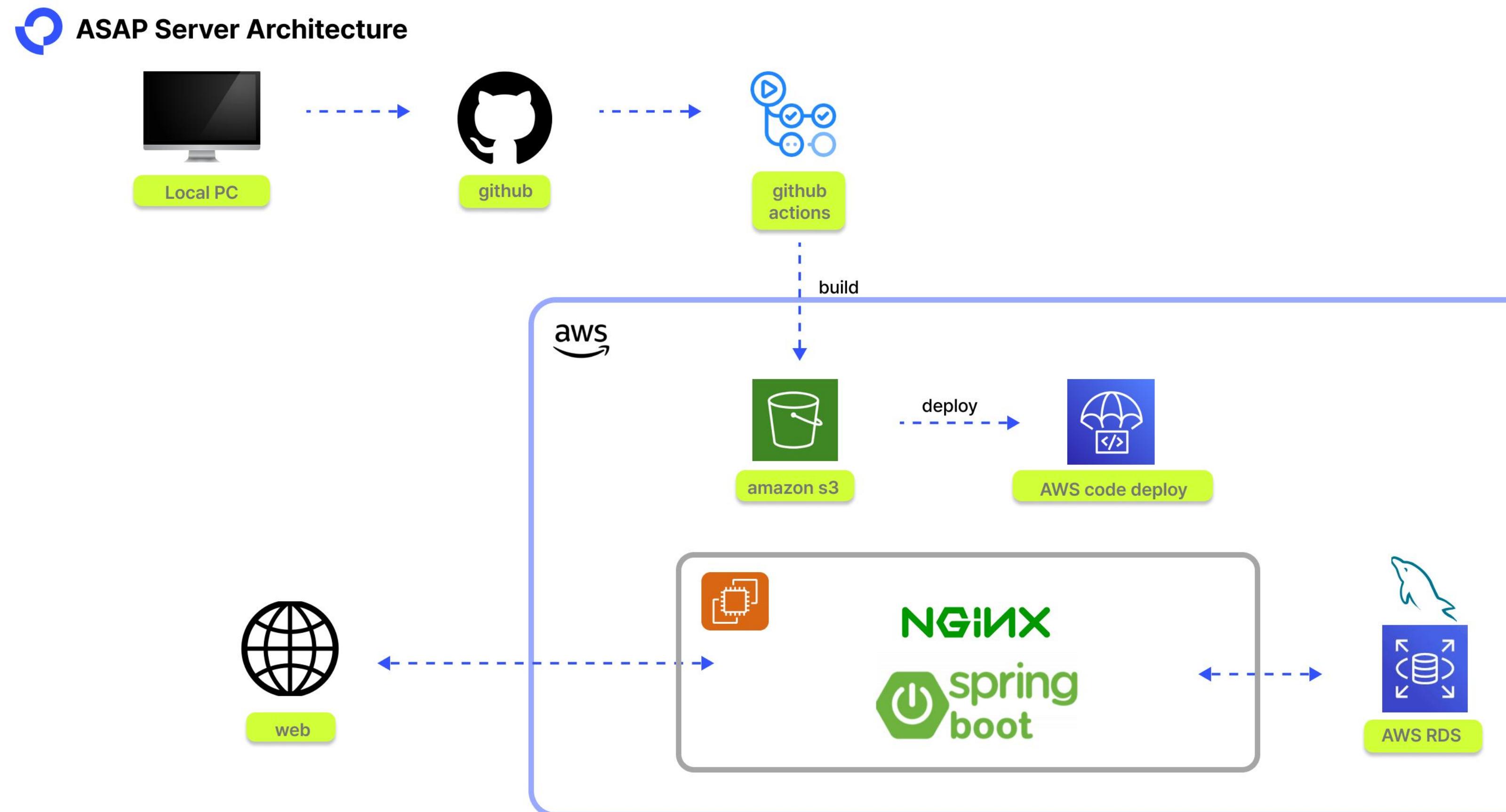
## 역할

- Github Actions, AWS Code Deploy CD 파이프라인 구축
- @SI4fj와 Logback을 활용한 Request Body 로깅  
관련 링크 : <https://velog.io/@devlynny/Spring-AOP>
- Spring Boot 기반 API 작성
- 개발 서버 배포 서버 분리
- Nginx를 이용해 리버스 프록시 구축
- Slack API를 통해 서버 장애 모니터링

# Projects

## ASAP <https://www.beginwithasap.com>

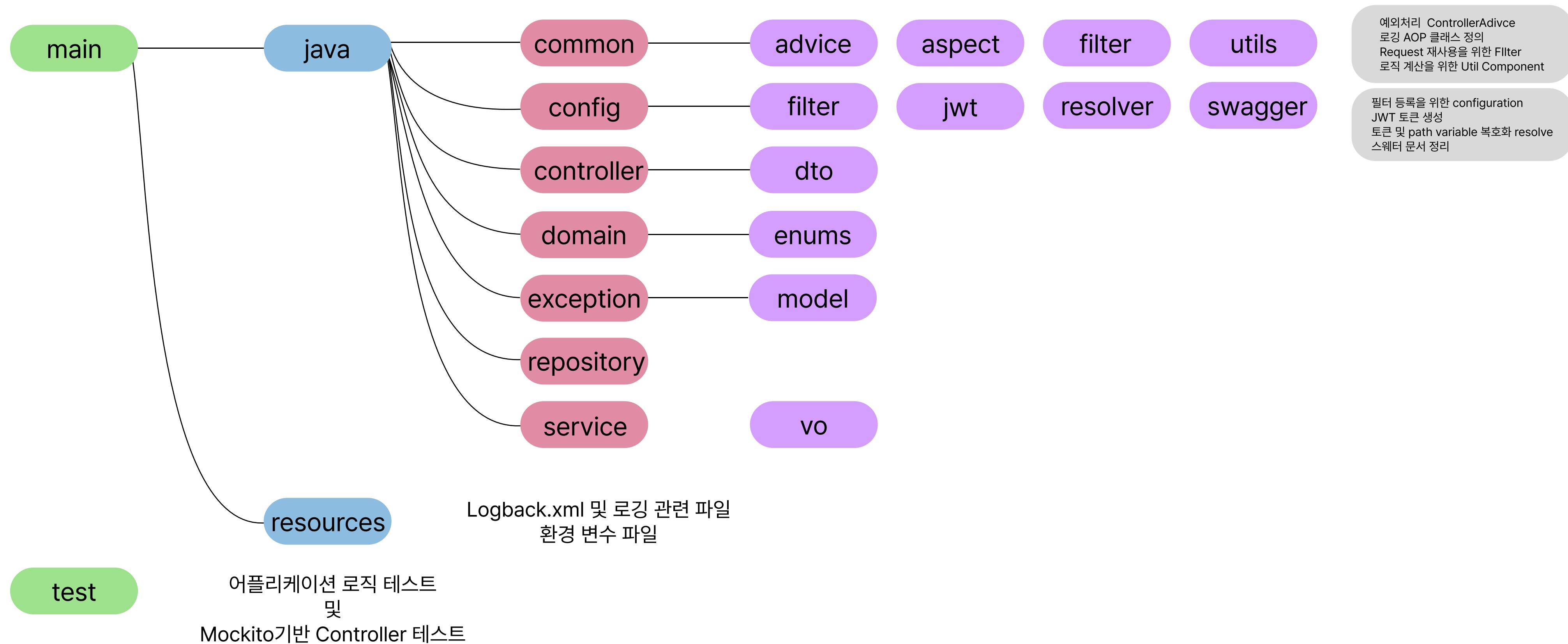
서버 아키텍처



# Projects

## ASAP <https://www.beginwithasap.com>

### 폴더 구조



# Projects

---

## ASAP <https://www.beginwithasap.com>

### 구현 상세

Spring AOP , Logback 사용하여 Request 로깅

#### 로깅 도입 배경

Request Body에 유효하지 않은 값을 확인해주기 위해 @Validation 어노테이션을 사용했지만 유효성 검사에 사각지대가 존재하였습니다. Request Body가 빈 값으로 전해져 서버 내부 오류가 발생하였고 서버 인스턴스 내부에 들어가 nohup.out 파일 내부를 확인하는 불편함이 있었습니다. 그의 경우 팀원과 함께 일관된 정보를 확인해야한다는 보장이 있어야 했기 때문에 파일을 생성하여 날짜별로 기록하는 방식을 사용했습니다.

#### 로깅 구현 방식

LoggingAspect 클래스

Controller 메소드가 작동될 때 로깅이 적용되기 위해 PointCut 정의  
Request에 대한 Response 또한 기록해야하므로 Around(메소드 호출 전 후에 수행) 적용

#### 문제 상황

단순히 Spring 내부에서 로깅할 경우, 외부 영역에서 요청이 들어오면서 오류가 난다면 요청 로그를 남기지 못한다.

HTTP에 접근하기 위해 ContentCachingRequestWrapper를 상속받아 Logging Filter를 만들어 Request Body 여러번 참조(읽음) 가능하게 함  
필터 정의 후 HttpRequestConfig를 통해 필터를 등록

logback-spring.xml을 통해 로깅 관련 설정, 서버가 로컬에서 구동될 경우 console에 기록, dev에서 구동될 경우 파일에 기록하는 방식으로 xml 파일 정의

# Projects

---

## ASAP <https://www.beginwithasap.com>

### 구현 상세

Slack API 연동하여 서버 내부 오류 모니터링

#### 모니터링 도입 배경

서버 내부 오류가 발생했을 때, 세부 사항을 즉각적으로 확인할 수 없는 문제점을 발견하고, 내부 오류를 빠르게 모니터링 하는 방법으로 협업 툴인 Slack을 연동했습니다.

#### 구현 방식

ControllerAdvice를 통해 체크 예외인 Exception이 발생하는 경우, request와 예외 내용을 담아 에러가 발생한 시점에 알림을 보낸다.

전체 메시지가 담겨 있는 LayoutBlock을 생성하고 이를 바탕으로 Slack에 전송하는 Message 생성

#### 구현 효과

단순 클라이언트의 요청에 대한 에러 뿐만 아니라, 악의적인 해커의 반복적 요청에 대해서도 대응할 수 있었음

JWT 토큰을 이용한 방장 인증

#### 토큰 기반 인증 방식 도입 배경

회의의 방장으로 참여한 사람의 권한을 확인하기 위한 간편한 방법을 고안한 후에 JWT 기반의 인증 방식을 사용했습니다

#### 구현 방식

User 도메인의 식별값인 user\_id를 이용해 JWT 생성  
JWT 형식으로 들어온 헤더에서 Resolver를 이용해 JWT 복호화

#### 구현 효과

복잡한 방식의 유저 인증 방식이 아닌 JWT 토큰으로 일원화된 방식으로 방장 인증

#### 구현 목표

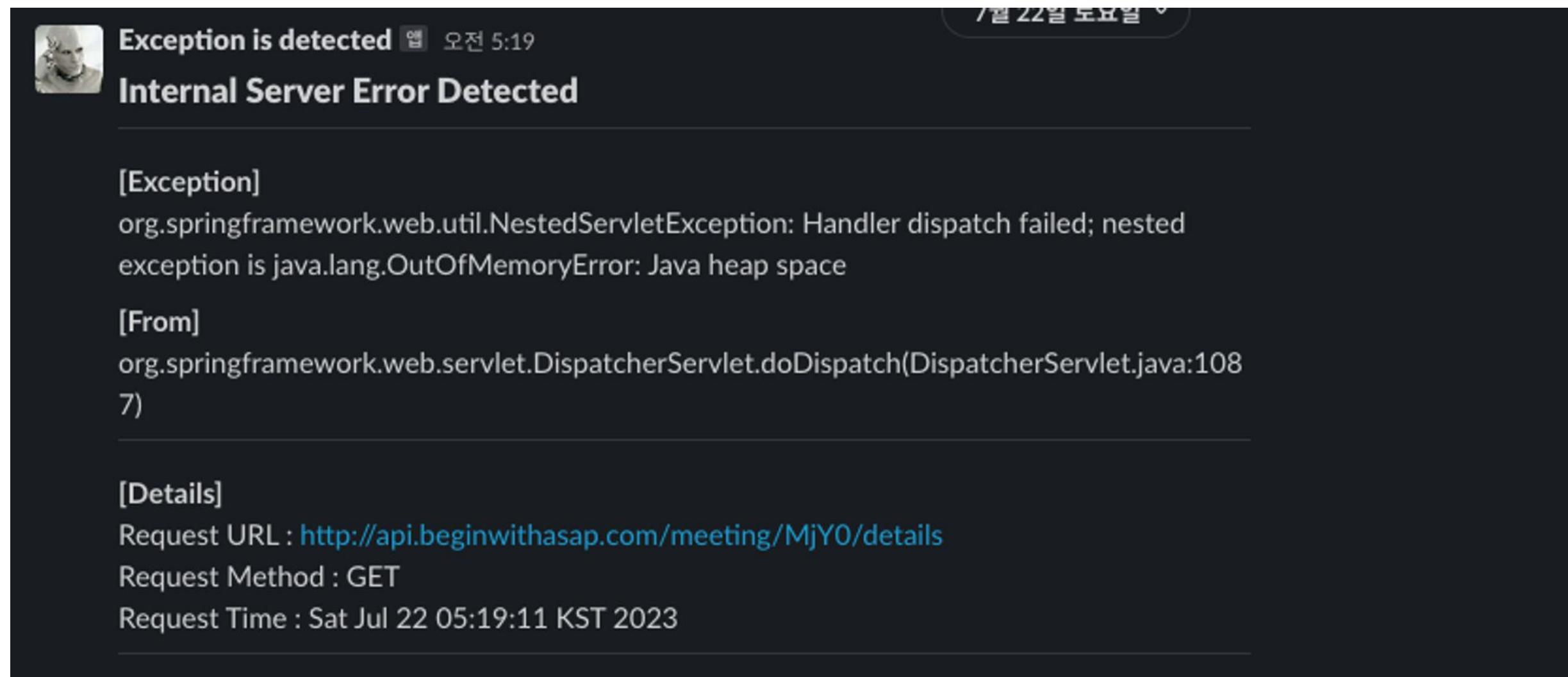
로그인 시 만료되지 않은 토큰이 존재함(유저의 쿠키에는 존재하지 않음)에도 불구하고 새로운 토큰을 만드는 리소스가 발생하는 것을 발견, 이를 위해서 Redis를 도입할 것을 검토 중

# Projects

## ASAP <https://www.beginwithasap.com>

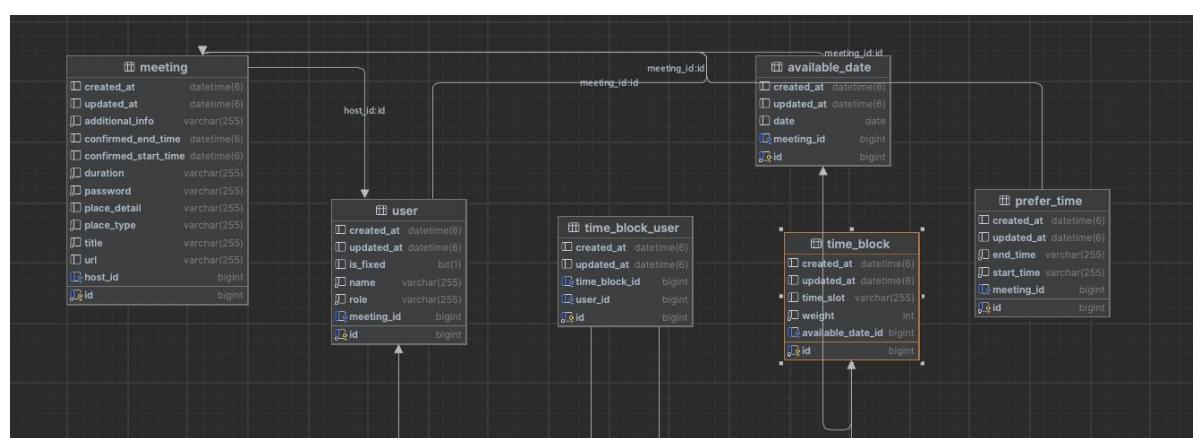
### 문제 상황

최적의 회의시간을 GET 하는 과정에서 Heap Space Out Of Memory 에러 발생



앞으로 회의 참여자수가 많아지면 언제든 발생할 수 있다고 판단하였고 로직을 변경하기로 했습니다.

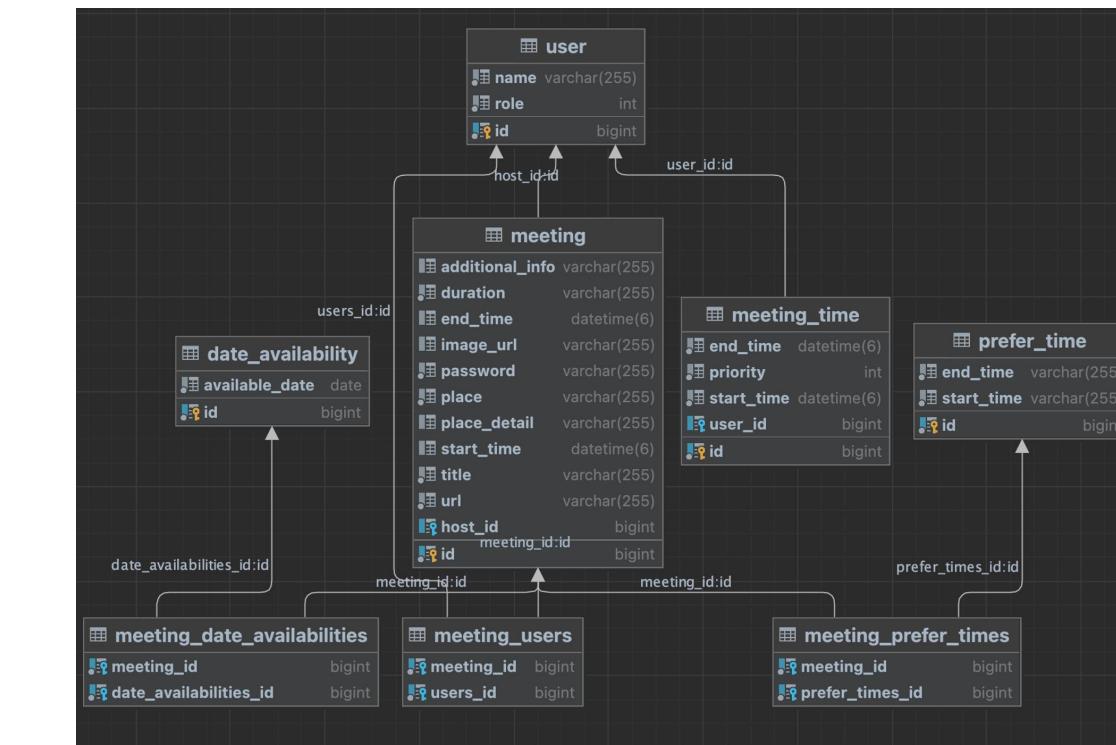
### 해결 방법 및 해결 후 ERD



엔드포인트는 유지하되 내부 DB 구조를 효율적으로 바꾸고자 했습니다. HashMap 등 객체를 가장 최소로 사용하는 방향을 고안하던 중 DB 관계에서 가능한 시간과 가능한 사용자를 저장하는 방법을 고안했습니다.

방장이 입력한 회의 날짜와 회의 진행 시간 범위에 해당하는 테이블을 만들고 그 각각 테이블에 참여가능한 회의 참여자 테이블의 연관관계를 구성하여 HashMap 등의 객체를 사용하지 않는 방법으로 해결했습니다.

### 기존 ERD와 로직



회의에 참여하는 참여자와 참여자의 가능시간을 담는 테이블을 두고 HashMap형식으로 <시간 , 가능한 사용자 List>를 찾아 가장 리스트가 큰 시간을 찾는 로직으로 구현했습니다. 이 때, 회의를 주체한 회의 방장이 제출했던 소요 시간을 고려하는 상황이었습니다. 이 로직과 더해 30분 단위로 시간을 잘라 구현했으므로 최대 252개의 HashMap이 발생하는 상황이었습니다.

# Projects

---

## ASAP <https://www.beginwithasap.com>

### 구현 상세

Interceptor, Redisson 사용한 동시 요청에 따른 예외 처리

### 구현 동기

회의 생성 후 참여자들의 가능 시간 입력에서 동시에 발생한 요청으로 중복된 자원 생성되는 현상을 발견했습니다.  
새로운 자원이 생기는 로직에 대해 대규모 WAS와 분리되어 동시에 온 요청을 막아야 할 필요성을 느꼈습니다.

### 구현 방식 링크 : [구현 회의](#)

구현 회의를 진행한 후 백엔드 팀원과 페어프로그래밍 형식으로 진행. 초기엔 setnx를 사용한 방식을 고민했지만, 시간이 지난 후 요청을 진행하는 것이 아닌 예외를 던져야 하기 때문에 Redis가 제공하는 Map을 사용

### 시나리오

1. Interceptor의 preHandle 메서드에서 회의 생성, 가능 시간 입력 API에 대해 Redisson Map을 사용하여 request가 온 IP와 Request Body값을 저장
2. putIfAbsent를 이용하여 이미 값이 존재한다면 429 Too Many Requests 예외를 던짐
3. afterCompletion 메서드를 활용하여 예외 발생 및 응답 제공 후 Map에서 해당 값을 지움

### 테스트

ExecutorService를 이용하여 쓰레드를 여러개 생성한 후 MockMvc로 동시 요청 시나리오 작성

1개의 요청을 제외한 나머지 요청이 429 TooManyRequests를 응답하는지 검증하는 방식으로 테스트 진행

개발자가 서비스의 가치에 부여할 수 있는 점을 생각하는 **백엔드 개발자 도소현**입니다.  
공유하는 문화 아래서 즐거움을 느끼며 성장합니다.  
사이드 이펙트를 줄이며 유지보수에 좋은 코드를 작성하게 위해 공부하고 있습니다.  
함께 성장하는 문화를 좋아합니다.

Github : <https://github.com/sohyundoh>

Contact : 010-6732-2315

E-mail : lyny.doh@gmail.com

**END.**