C;SAFY

시스템 기술서

목차

I. 개요

1. 프로젝트 개요

자율 프로젝트는 공통, 특화 프로젝트에서 갈고 닦아온 기술을 사용하여 자유롭게 서비스를 개발하는 것이 목표입니다. 어떠한 서비스를 제공할 것인지 기획하고 개발, 테스트, 배포하고 유지보수 하는 과정이 담겨있습니다. 본 기술서에서는 이번 프로젝트에서 그 목표 달성을 위해서 어떤 상황과 조건이 주어졌고, 프로젝트를 진행하면서 이를 어떻게 달성해 나갔으며 무엇을 보고 느꼈는지에 대해 정리해보고자 합니다.

2. 프로젝트 배경

개발자로서 취업하기 위해 꼭 공부해야 하는 것 중 하나는 바로 Computer Science (이하 CS) 지식입니다. CS 지식을 공부하기 위해 많은 취업 준비생들은 하염없이 인터넷 검색을 하고, 책을 사서 읽습니다.

이제는 스마트하게 공부하십시오. C;SAFY 는 모의고사를 통해 현재 부족한 CS 분야를 분석해줍니다. 방대한 CS 정보를 분야별로 제공하고, 실전 면접의 기출문제를 제공하여 실력 향상에 도움을 줍니다.

3. 프로젝트 핵심 기술

이번 프로젝트를 하면서 사용한 핵심 기술들입니다.

- **Spring Web**
  전반적인 Rest Controller 구현 및 작업환경을 구축하였습니다.

- **Spring Security**
  Spring Security 를 통하여 회원가입을 포함한 회원관리, 비밀번호 암호화, 해쉬 기능, jwt 유효 검사 등 필터 및 인터셉터 수행, cors, csrf 이슈 해결 등 보안을 포함한 인증, 권한을 관리하였습니다.

- **JWT**
  JSON Web Token 을 활용하고 Spring Security 와 연계하여, 안전하고 편하게 회원 정보 교환을 수행하였습니다.

- **OAUTH**

  SNS 서비스로서 회원의 진입 장벽을 낮추고, 가볍게 진입할 수 있는 환경을 구축하였습니다.

- **Validation**

  대상 정보가 유효한지 파악하고 관련 오류를 제어하고, message_xx.yml 설정과 연계하여 국제화 메시지를 구현하였습니다.

- **Spring Data JPA**

  Spring 에서 제공하는 ORM 인 Hibernate 를 사용하여 DB 에 종속되지 않고, 객체 중심의 개발을 할 수 있었고, Spring Data 를 통하여 생산성은 물론 유지/보수 등의 관리도 하기 편한 환경을 구축하였습니다.

  또한 Fetch Join 기능을 활용하여 현업에서 발생할 수 있는 '1+N' 이슈 등을 사전에 해결하고 효율적인 쿼리셋을 작성할 수 있었습니다.

- **Lombok**

  코드 자체의 가독성을 올리고, Getter 나 setter 을 많이 활용하는 DTO 나 의존성 주입이 필요한 클래스들에게 RequiredArgsConstructor 등의 기능을 제공하여 코드를 간략하게 짤 수 있었습니다.

- **MySQL MariaDB**

  관계형 DB 인 MariaDB 를 메인 DB 로 사용하였습니다.

- **AWS (EC2 + RDS)**

  EC2 인스턴스에서 서버를 구축하고, RDS 서버를 이용하여 관계형 DB 를 적재하였습니다.

- **AWS S3**

  S3 에 프로필 이미지 등을 업로드할 수 있게 static 한 이미지 서버를 구축하였습니다.

- **React.js**

  프론트의 Javascript 라이브러리로 React.js 를 활용하였습니다.

- **Recoil**

  전역 상태인 store 을 관리하기 위해 recoil 을 사용하여 action 과 reducer 을 구축하였습니다.

- **MUI**

  React의 UI 라이브러리로 유명한 Material UI와 styled component를 필요한 상황에 맞게 활용하여 디자인하였습니다.

- **Unity**

  메타버스를 구축하기 위한 게임 엔진입니다. Unity 특유의 가벼움과 용량이 웹 환경에 적합하다고 생각하여 기술 스택으로써 Unity를 채택하였습니다. 사용언어는 C# 입니다.

- **WebGL**

  Web Graphic Library로 캔버스에 3D 그래픽인 Unity를 렌더링 하기 위한 javascript API 입니다. Send와 JsLib를 사용하여, react와 Unity 간의 상호소통을 구축하였습니다.

- **Photon Network**

  다양한 네트워크 환경을 제공하며, 유니티 네트워크에서 자주 사용되는 대중적인 네트워크 솔루션입니다. 동기화와 메치메이킹, 방 권한과 Websocket 등을 통한 채팅 기능 제공 등을 제공합니다.

- **Let's Encrypt**

  SSL 무료 인증서를 받아 웹 서버에 https를 적용하고, 통신에 보안을 적용하였습니다.

- **Kakao Pay**

  Kakao API로 제공되는 카카오 결제 시스템을 이용하여, 프리미엄 회원 등의 상품을 판매하고 결제할 수 있게 하였습니다. 결제는 프론트에서 진행되지만 그 내역은 토큰으로 전달되어 백 서버를 거쳐 DB에 저장합니다.

4. 비즈니스 모델

코로나 19 시대 몇 년 사이에 IT 구직자의 수요가 폭증하고 있고, 그와 동시에 관련 서비스도 성장하고 있습니다. 예를 들어 코딩 연습, 취업 중개 플랫폼으로 알려진 프로그래머스나 AI 시험 감독 서비스 모니터의 모회사인 그렙은 68 억의 시리즈 A 급 투자금액을 유치하는데 성공하였습니다.

-그림 2021 년 상반기 IT 구직자 폭증

저희는 이렇게 폭증하는 개발자 공급/수요환경에서 더 좋은 개발자를 뽑고 싶어하는 욕구와 더 훌륭한 학습 환경을 원하는 개발자의 욕구를 만족할 수 있는 서비스가 없다는데에 주목하였고, 빠르게 커지는 시장 자체에 주목하여 이번 교육 플랫폼을 만들게 되었습니다.

프로젝트에 있어서 비지니스 모델을 파악하고 구축해봄으로써, 저희는 사용자가 원하는 가치와 우리가 제공할 수 있는 가치에 대해 진지하게 고려해볼 수 있었고, 프로젝트가 단순히 코드나 Hello world 를 구현하는 정도로 끝나지 않고, Real world 를 고려하고 사용자의 욕구를 반영한 하나의 product 를 구성할 수 있었습니다.

1. 문제	4. 솔루션	3. 가치 제안	9. 경쟁우위	2. 고객 세그먼트
1) 코딩 테스트, AI면접 등 다른 입사 준비에 비해 주목 받지 못하는 기술면접. : 프로그래머스, 백준, 코드잇 등 코딩 연습을 서비스 하는 업체나 AI면접등 일반 면접, 태도를 도와주는 업체는 다수 있으나 이상할 정도로 없는 기술면접 사이트. 2) CS 데이터를 개인이 정리한 사이트는 간혹 있으나 그것을 점검 할 수 있는 수단이 거의 없음. : 기술 블로그, 깃허브처럼 개인이 공부하기 위해 혹은 개인 사전처럼 쌓아놓는 곳은 많지만, 쌓여있는 데이터를 내가 확인하고 실력을 자체 점검 할 수 있는 구조는 아님. 3) 정보처리기사, 빅데이터 자격 처럼 기술시험을 준비할 수 있는 문제집은 있으나 접근성과 방향성에서 괴리감을 가짐. : 기술면접과 공인자격시험은 결을 달리하는 경우가 많아 대체하기 적절하지 않으며 연습 또한 온라인이 아닌 서적으로 밖에 공부 할 수 있는 방법이 없다.	1) 기술 면접 특화 서비스를 통해 개발자 면접 준비에 있어 부족한 영역을 독자적으로 구축. 2) 시험과 피드백, 점수 통계 데이터 제공 등으로 단순 정보의 제공을 떠나 학생의 역량 향상에 중점을 둠. 3) 여러 CS지식의 활용처 중 '기술면접' 으로 집중 타겟을 세팅하고 접근성 좋은 '웹 서비스' 로 개발해 방향성과 접근성의 두 핵심 경쟁력을 확보.	비전공자 출신 학생들에게는 투자 해야하는 '긴 공부 시간', '중분한 비용', '작은 시장으로 인한 낮은 접근성' 세 가지가 장애물로써 준비단계부터 많은 어려움이 따름. 이를 위해 '핀포인트식 고효율 학습법', '낮은 비용', '높은 접근성' 을 통해 기존 학생들의 요구사항을 일거에 만족 시킬 수 있음. 관련 사이트나 정보원이 정보/점검으로 분리 되어 있고 특히 점검 쪽이 부실한 상황에서 이 두 가지를 한번에 해결 할 수 있는 서비스의 존재는 학생들에게 있어 대체 불가능이며 반드시 필요함.	'시장 선점' 흔히 입사준비의 시험들에는 인적성 검사, 코딩 테스트, 기술 면접, 일반 면접 등이 있으나 이중 기술 면접만큼은 시험에 특화되어 공부 할 수 있는 서적도 DB도 서비스도 존재 하지 않음. 정보처리기사 출제문제를 온라인에서 풀게 하는 서비스는 작게나마 존재 했으나 문제 3에서 보듯 구직자와는 다소 맞지 않는 방향성 때문에 완전한 경쟁자가 되긴 어려움. 또한 객관적 지식이 주요 상품인 만큼 미리 시장을 선점할수 있다면 후발 주자가 등장해도 고객의 이탈은 크지 않을것이라 판단됨.	지식은 있지만 테스트로 실력을 확인해 보고 싶은 학생들. 기술면접 연습을 직접 해보고 싶은 학생들. 모자란 부분만 시원하게 지적받고 피드백 받고 싶은 학생들. 장황하게 설명되어 있는 각종 블로그식 대신 빠르고 집중적으로 개념학습을 하고 싶은 급한 학생들 등 개발자 공부를 시작한지 얼마 안되어 이론적 배경이 부족한 신입 개발자들 뿐 아니라 이론 점검을 원하는 학생들 중 기술면접을 필히 준비해야 하는 모든 학생들.
	8. 핵심지표		5. 채널	
	회원 수. 프리미엄 회원 수. 프리미엄 계정 재구매율, 평균 유지기간. 회원 당 단위 기간(주/월) 당 푸는 문제 량. 회원당 체류시간(분/시간), 단위 시간당 문제풀이 량. 집중 접속 날짜 구간.		프로그래머스, 자소설 닷컴, 로켓펀치 등 개발자 밀도가 높은 사이트 광고. 주 홍보 채널로는 취준생들이 자주 다니는 취업카페 혹은 포털을 메인으로. 전문 개발자들보다는 초년생, 신입 개발자들이 검색해서 발견하기 쉽도록 전문적인 사이트 보단 대중적인 사이트를 통한 홍보에 주력한다.	

7. 비용 구조	6. 수익 흐름
초기 데이터 베이스의 질적, 양적으로 충분한 데이터 확보를 위한 인원, 전문가 인건비. 시장 서비스를 선점 하기 위한 공격적 마케팅 비용. 시스템 구축 및 유지를 위한 개발, 보수, 유지 비용.	일반 계정, 프리미엄 계정 차등화. 다른 학습 사이트와 연계하여 고객 유치, 광고 유치. CS 문제집 등 관련 업계 광고 유치. 비대면 기술 면접 서비스로 확장할 시 프로그래머스처럼 기업 제휴도 가능.(미래)

-그림. 스타트업에서 쓰이는 린캔버스를 통한 BM 설계

II. 핵심 기술 소개

1. Spring Data JPA

기존의 Mybatis 같은 프레임 워크를 통한 DB 중심의 쿼리 질의에서, JPA 를 이용하여 객체 중심의 개발이 이루어질 수 있도록 구성하였습니다.
코드로 쿼리를 작성하기 때문에, 컴파일 시점에 오류를 확인할 수 있었습니다. JPA 는 기본 메소드를 제공해주어 기본 쿼리에 대해서는 빠른 코딩을 할 수 있도록
도와주었고 Pageable 을 이용하여 무한 스크롤을 쉽게 구현할 수 있도록 기능을 제공 하였습니다.

Fetch Join 을 이용하여, N+1 이슈를 해결하였고, Entity 내 여러 개의 자식엔티티를 가져와야 하는 경우 `hibernate.default_batch_fetch_size` 옵션을 이용하여 기존 조건절을 In 절로 묶어 조건절 비교값이 다른 쿼리들을 한 번에 처리할 수 있도록 설계하였습니다.

```
// 최근 본 강의
@Query("select v FROM VideoPlay p join fetch Video v ON v.id = p.video.id WHERE p.userSeq =:userSeq order by p.playAt DESC")
Page<Video> findUserStudy(@Param("userSeq") Long userSeq, Pageable pageable);
```

- 그림 fetch join 을 이용한 N + 1 이슈 해결 및 Pageable 적용

```
Hibernate:
    select
        interviewlO_.interview_seq as intervie3_6_1_,
        interviewlO_.id as id1_6_1_,
        interviewlO_.id as id1_6_0_,
        interviewlO_.interview_seq as intervie3_6_0_,
        interviewlO_.user_seq as user_seq2_6_0_
    from
        interview_likes interviewlO_
    where
        interviewlO_.interview_seq in (
            ?, ?, ?, ?
        )
```

-그림 `hibernate.default_batch_fetch_size` 를 이용하여 쿼리가 `in` 절로 묶인 모습

2.Spring Security (+OAUTH, JWT)

현 프로젝트가 기존의 구글, 네이버와 같은 보안이 검증된 사이트 보다 보안이 더 우수하다고 볼 수 없기 때문에, 로그인에 OAUTH2 를 적용하여 보안성을 올릴 수 있도록 설계하였습니다. 특히, 여러 OAUTH 인증 방식 중에서 Authorization Code Grant 인증 방식을 사용하여 Access Token 을 직접적으로 Front-End 에 전달하지 않고 Backend 에서 Access Token 을 검증한 후 JWT Token 을 만들어 전달해 주면서, Access Token 이 탈취되지 않도록 구성하였습니다.

Jwt Token 은 SecretKey 를 이용하여 builder 를 통해 토큰에 저장하고 싶은 데이터들을 넣고, 토큰 만료시간을 지정한 후 HS256 을 이용하여 발급했습니다. HS256 대칭키 암호화 알고리즘을 사용하여 해시 값의 특징을 활용할 수 있게 설계하였고, 이를 통해 메시지 변조여부를 확인할 수 있도록 설계하였습니다.

```
// JWT 토큰 생성 (parameter List<String> roles 는 사용 안해서 제외)
public String createToken(String username, Long id) {
    Claims claims = Jwts.claims().setSubject(username); // JWT payload 에 저장되는 정보단위 (sub)
    claims.put("user_seq", id);
    claims.put("username", username);
    Date now = new Date();
    return Jwts.builder()
            .setClaims(claims) // 정보 저장
            .setIssuedAt(now) // 토큰 발행 시간 정보
            .setExpiration(new Date(now.getTime() + tokenValidTime)) // set Expire Time
            .signWith(SignatureAlgorithm.HS256, secretKey)  // 사용할 암호화 알고리즘과 signature 에 들어갈 secret값 세팅
            .compact();
}
```

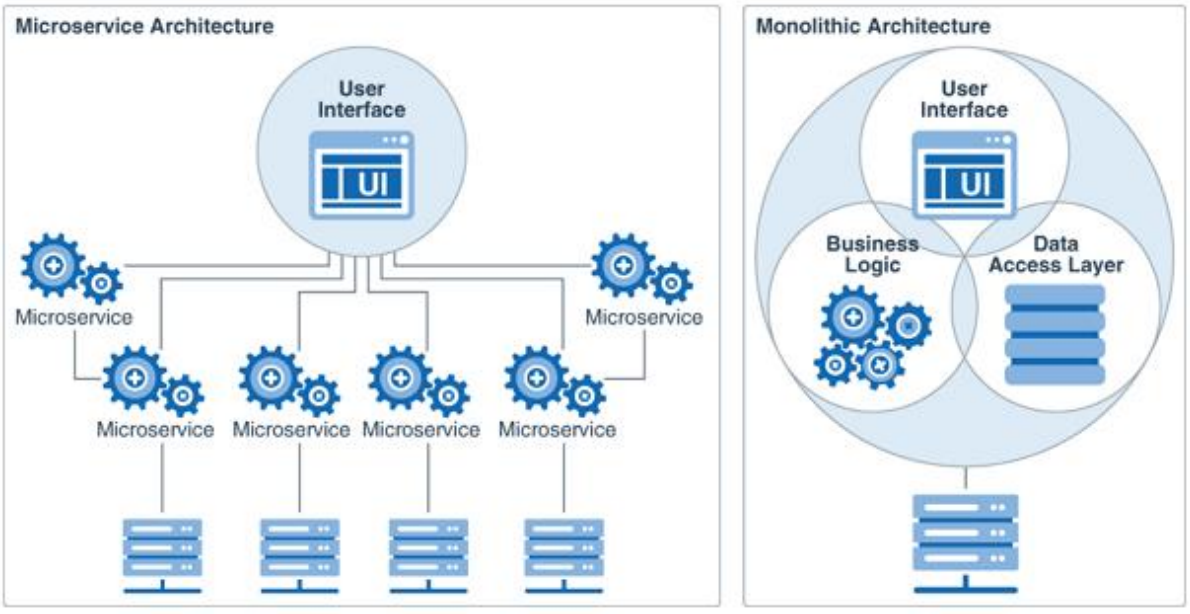
- 그림 JWT Token 생성

3. MicroServices Architecture (이하, MSA)

과거에는 소프트웨어의 모든 구성요소들이 하나의 Application 에 통합 개발되어 빌드 되고 배포하는 설계가 많았습니다. 이러한 방식을 모놀리식(Monolithic) 아키텍처라고 부릅니다. 그러나, 모든 기능 및 DB 가 통합되어 있다 보니 다음과 같은 여러가지 한계점이 드러나게 되었습니다.

부분적인 장애가 전체 시스템의 장애로 이루어질 수 있던가, 결합도가 강해 특정 부분의 수정이 다른 부분에 영향을 미치거나 연계된 부분이 발생하게 되던가, 클라우드 환경에서 스케일 아웃이 힘들고 인스턴스 자동 확장 등이 제한적이게 되고, 개발자 입장에서도 배포나 생산성적인 면에서 문제가 발생하였습니다.

이러한 단점을 극복하기 위해 기업에서 활발하게 도입하는 것이 수많은 기능 단위로 서비스를 분산하여 배포하는 Micro Services Architecture 입니다. 구축은 힘들지만 구축한 이후에는 모놀릭 아키텍쳐에서 발생하던 문제점을 해결하는 동시에 장애 격리, 독립 무중단 배포, DB 분리, 작은 서비스 단위의 개발 테스트, 운영 배포로부터 오는 높은 생산성 등의 추가적인 이득을 기대할 수 있습니다.



- 그림 MSA 와 모놀리식 아키텍처 구성 차이점

다만, 위에서 말한 것처럼 첫 도입이 복잡하고, 데이터베이스의 트랜잭션을 위해서 별도의 메시징 큐를 운용해야 하는 등, 모놀리식 아키텍쳐에서는 생각하지 않아도 되는 문제를 고려해야 한다는 이슈가 발생합니다.

저희는 Netflix 의 노하우가 담긴 Spring Cloud 디펜던시를 사용하여 그나마 쉽게 MSA 를 구성할 수 있었습니다. 이번 프로젝트에서는 실제 MSA 를 위한 실습 MSA 의 동작을 학습하고 응용하는 것을 목적으로 백 서버를 구축했습니다. 목적 달성을 위해서 넷플릭스가 제공하는 오픈소스인 디펜던시를 활용하여 핵심적인 기능을 구현함으로써 전체적인 흐름을 파악하는데 집중하였습니다.

4. Spring Cloud 를 통한 MSA

Spring Cloud 는 Spring Boot 기반으로 제작되었으며, 클라우드 환경을 보다 편리하게 구축하기 위해 설계된 라이브러리입니다. 이러한 Spring Cloud 에서는 MSA 구축에 널리 쓰인 Netflix OSS 프레임워크의 핵심적인 라이브러리를 포함하고 있습니다.

Spring Cloud 는 분산 시스템 구성과 애플리케이션을 개발하기 위해 필요한 개발 환경과 서비스, 그리고 개발이나 구성에 관련된 패턴을 제공함으로써 Cloud Native Architecture 를 위한 Microservice 애플리케이션을 빠르게 개발할 수 있도록 제공하는 개발 도구이자, 개발 Platform 이고 이를 위해서, 수많은 환경과 옵션을 제공합니다. 이번 프로젝트에서 저희 프로젝트에 맞는 형태의 MSA 를 구축하는 것도 중요하지만, 학습 차원에서 가장 최소한의 설정을 한 기본적인 형태의 MSA 를 구축하였습니다.

우선 오토 스케일링 등 내부 IP 가 동적으로 변하거나 생성될 수 있기 때문에, 분산되어 있는 서비스를 호출할 때 서비스의 위치를 알아낼 수 있는 서비스 디스커버리(Service Discovery) 기능이 필요한데 저희는 이 기능을 Spring Cloud 에서 제공하는 Eureka 를 이용하여 구현하였습니다. 디스커버리 서버에 Eureka Server 를 탑재하고, 각각의 서비스에 Eureka Client 를 탑재한 후, Eureka Server 에 Eureka Client 를 등록함으로써, 서비스들의 고유 ID 를 매핑하여 관리하고, 조회하여 요청시 Gateway 에 전달할 수 있게 됩니다.

로드 밸런싱 등의 기능이 포함된 API Gateway 의 구현은 마찬가지로 Spring Cloud 에서 제공하는 Spring Cloud Gateway 를 통해 구현되었습니다. 기본적인 로드 밸런싱을 통해서 요청이 들어왔을 경우 어느 인스턴스로 이동해야 할지 결정해주어 부하를 분산하고, 이동해야 하는 서비스를 등록하고 클라이언트의 요청에 맞는 서비스로 연결해주는 동시에, 공통 Filter 전용 Filter 등의 역할을 수행할 수 있습니다. Filter 를 통해 인증 및 권한을 부여하거나 서비스 검색 통합 등의 작업을 수행할 수 있으며 화이트 리스트, 블랙 리스트 등을 관리할 수 있습니다.

외부 클라이언트에서 접근시 Flow 를 정리하자면, 외부 요청시 단일 지점인 Gateway 가 각 서비스의 정보를 Eureka 에게 물어보고, 해당 서비스를 호출하고 응답을 받아 외부 클라이언트에게 답해주는 형태로 시스템 내부 구조를 숨기고, 외부 요청에 대한 적절한 응답을 해줄 수 있게 됩니다.

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CHAT-SERVICE	n/a (1)	(1)	UP (1) - chat-service:d12114378344ed03ca4cfc1d908ae6ab
CS-SERVICE	n/a (1)	(1)	UP (1) - cs-service:8740271ccd15c26d9dfb66aef9dc6b6a
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - ip-172-26-2-107.ap-northeast-2.compute.internal:gateway-service:8000
PAY-SERVICE	n/a (1)	(1)	UP (1) - pay-service:4efb505353f786fa219d6ec2750b8907
USER-SERVICE	n/a (1)	(1)	UP (1) - user-service:f8817f66f860f50ce780789948c5d3aa

- 그림 Netfilx Eureka Service Discovery

```
- id: user-service
  uri: lb://USER-SERVICE # lb => load balancer
  predicates:
    - Path=/user-service/actuator/**
    - Method=GET,POST
  filters:
    - RemoveRequestHeader=Cookie
    - RewritePath=/user-service/(?<segment>.*), /$\{segment}
```

- 그림 로드밸런싱 및 메소드, URL 별 필터 적용

```
eureka:
  instance:
    instance-id: ${spring.application.name}:${spring.application.instance_id:${random.value}}
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: htt[redacted]/eureka
```

- 그림 random InstanceID 를 통한 인스턴스 랜덤화

5. 마이크로 서비스간 통신 FeignClient

마이크로 서비스 간에도 통신이 필요한 경우가 있습니다. MSA 의 단점이 이 마이크로 서비스 간의 통신에 의한 성능 저하이고 필요하다는 점이고 이를 위해 gRPC 등 여러가지 최적화 방안을 고려하고 있지만, 저희는 가장 기초적인 MSA 를 구현한다는 목적에 충실하기 위해 OpenFeign 에 의한 REST 통신을 사용하였습니다.

OpenFeign 을 사용하기 위해서는 Rest call 을 추상화한 Spring Cloud 의 라이브러리인 FeignClient 를 사용해야 하고, 개발자는 직관적으로 마치 마이크로서비스 내에 포함된 함수나 서비스를 사용하는 것처럼 편안하게 함수를 사용할 수 있습니다.

FeignClient 를 사용하기 위해서는 관련 라이브러리 설치 후, main 클래스에 @EnableFeginClients 선언 후에나 인터페이스를 통해 구현할 수 있으며, 이 인터페이스를 주입함으로써, 애플리케이션 내의 함수를 사용하듯이 해당 함수를 사용할 수 있습니다.

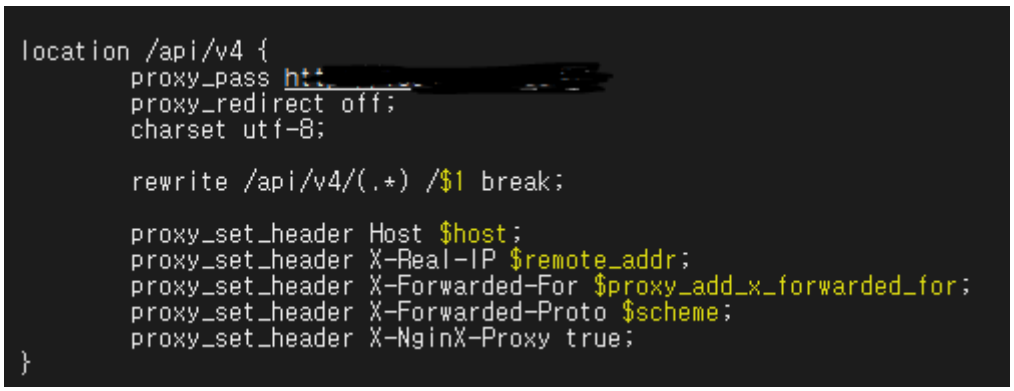
```java
@FeignClient(name="user-service", url = "http://localhost:8000/user-service")
public interface UserServiceClient {

    @GetMapping("/tokenvalidated")
    String checkTokenValidated(@RequestParam("inputToken") String inputToken);

    @GetMapping("/token/user")
    UserDto getTokenUser(@RequestParam("inputToken") String inputToken);

    @GetMapping("/email/seq")
    UserDto getUserSeqOnEmail(@RequestParam("email") String email);

}
```

- 그림 타 서비스와의 서비스간 통신 (FeignClient)

6. MSA 인증 서버

인증이란 특정 계정에 대한 소유를 확인하는 과정으로 흔히 ID 나 Password 를 사용한 로그인이나 OAuth 를 통해 이루어집니다. 본 프로젝트에서는 인증만을 위한 서버를 별도로 구축하였습니다. 이를 Authorizer 이라고 부르며, 이 이후 발행되는 정보로 마이크로 서비스 내에서 인가할 수 있게 하여, 토큰의 만료, 유효, 권한 확인 등을 진행하였습니다.

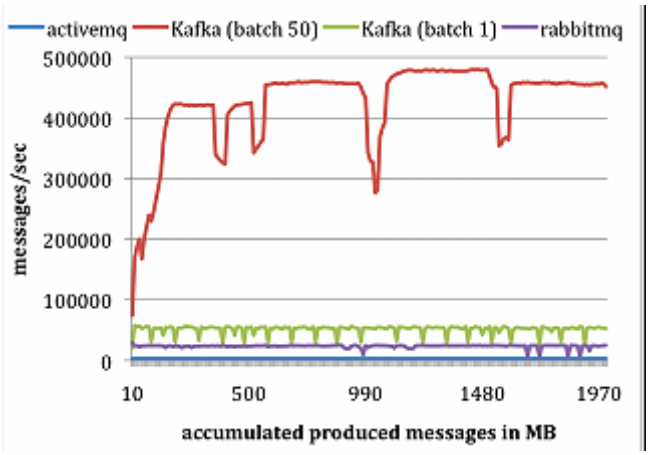
이렇게 구축할 경우 차후 Payload 추출에 관해서 인증서버는 어떠한 신경을 쓰지 않아도 되고 Resource 서버 쪽에서도 애플리케이션 요청이 매우 편해지는 장점이 있어서 개발 효율성이 올라가게 되고, 이번 프로젝트에서 사용한 JWT 토큰과 매우 궁합이 좋습니다.



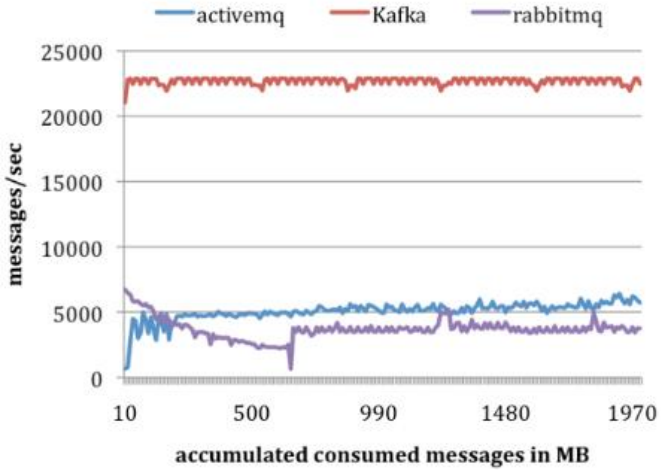
- 그림 인증서버 분리 (NGINX 설정)

7. Spring Cloud Bus

Spring Cloud bus 를 통해서 마이크로 서비스에서 경량 메시지 브로커와 연결되고 상태 및 구성에 대한 변경사항을 연결된 마이크로 서비스에 전달(Broadcast)할 수 있습니다. 이때 사용할 메시지 브로커로 크게 AMQP 와 Kafka 두 종류가 있는데 AMQP 란 Advanced Message Queuing Protocol 의 약어로 RabbitMQ 등으로 대표됩니다. 메시지 지향을 하는 만큼 신뢰성이 보안이 높지만 속도가 느리고(초당 20 건) 안전하게 보내는 것이 목표이기 때문에 상대적으로 보안이 낮지만 속도가 빠른(초당 10 만건) 분산형 스트리밍 플랫폼인 Kafka 를 선택하였습니다.



- 그림 Kafka 와 타 메시지 브로커 Consume 속도차이



- 그림 Kafka 와 타 메시지 브로커 Produce 속도차이

8. Kafka 를 통한 메시지 브로킹

메시지 브로커란 메시지 처리 또는 메시지 수신자에게 메시지를 전달하는 시스템이며, 독립된 애플리케이션 간에 데이터를 주고 받을 수 있도록 하는 디자인인 MOM (Message Oriented Middleware, 메시지 지향 미들웨어)을 기반으로 구축되어 있습니다.

서비스간 결합도가 낮은 MSA 에서는 데이터 송수신을 비동기로 처리하는 Message Queue 를 사용하는 게 효율적이고, 저희는 대표적인 메시지 브로커인 Kafka 를 사용하였습니다. Kafka 란 Apache 에서 만든 Pub/Sub 구조에 특화된 메시지 큐 시스템으로, 대용량의 데이터를 실시간으로 처리하는데 특화되어 있습니다. 메시지를 파일 시스템으로 저장하여 메시지가 유실될 우려가 적으며, Consumer 가 Broker 에 Pulling 방식으로 메시지를 가져오기 때문에 Consumer 가 처리할 수 있는 메시지만 가져올 수 있습니다.

카프카의 브로커는 토픽을 기준으로 메시지를 관리하는데, 메시지를 보내는 측에서 토픽이라는 메시지 저장소에 메시지를 저장하고, 메시지를 가져가는 측에서 원하는 토픽에서 메시지를 가져가서 처리하게 됩니다.

- 그림 카프카 Topic 을 통해 전달되는 메시지

9. Kafka Connect 구현

카프카는 확장성(scale-out)과 고가용성(high availability)을 위하여 broker 들이 클러스터로 구성 되어 동작하도록 설계되어 있습니다. 심지어 broker 가 1 개 밖에 없을 때에도 클러스터로써 동작하는데, 이런 클러스터 내의 broker 에 대한 분산 처리는 브로커 관리자인 Apache ZooKeeper 가 담당하게 됩니다.

Zookeeper 를 통해서 클러스터 최신 설정정보 관리, 동기화, 리더(Controller) 채택 등 클러스터 서버들이 공유하는 데이터를 관리할 수 있게 되고, Broker 에 분산 처리된 메시지 큐의 정보들을 관리하고 Zookeeper 없이는 Kafka 구동이 불가능합니다.

Kafka Connect 는 Data 를 Import/ Export 할 수 있게 돕는 툴입니다. 코드 없이 Config 설정 만으로 데이터의 이동을 지원하고, 커스텀 Connector 등을 동해 다양한 Plugin 을 제공합니다. 기존 DB 에서 가져오는 쪽을 Connect Source, 새로운 DB 로 보내는 쪽은 Connect Sink 라고 합니다.

저희는 이번 프로젝트에 이 Zookeeper 를 2181 번 포트, Kafka Server 를 9092 번 포트, Kafka Connector 을 8083 번 포트에 구동하고, Zookeeper -> Kafka Server -> Kafka Connect 순으로 잇고 Sink Connector 를 이용하여 DB 데이터 조정을 구현하였고, Source Connector 를 연계하여 DB 간 데이터 동기화를 구현하였습니다.


```
{
    "name": "user-sink-connect",
    "config": {
        "connector.class": "io.confluent.connect.jdbc.JdbcSinkConnector",
        "delete.enabled": "false",
        "connection.password": "[redacted]",
        "auto.evolve": "true",
        "connection.user": "S06P31A102",
        "tasks.max": "2",
        "topics": "user",
        "name": "user-sink-connect",
        "auto.create": "true",
        "connection.url": "jdbc:mysql://stg-yswa-kr-practice-db-master.mariadb.database.azure.
            com:3306/S06P31A102"
    },
    "tasks": [
        {
            "connector": "user-sink-connect",
            "task": 0
        },
        {
            "connector": "user-sink-connect",
            "task": 1
        }
    ],
    "type": "sink"
}
```


- 그림 KAFKA Sink Connector 적용 (Spring <-> DB)

10. Unity – WebGL 을 통한 메타버스

유니티는 게임엔진으로써 가벼운 것이 특징이고 3 억명이 넘는 유저를 보유한 제페토 메타버스에서도 실제로 사용되는 엔진입니다. 이번 C;SAFY 프로젝트에서는 여럿이서 같이 공부하고 싶어하는 사용자의 니즈를 충족시키기 위한 학습실 공간, 공부와 게임을 결합시킨 OX 퀴즈 서바이벌 게임 공간, 그리고 실제 면접장에 와있는 듯한 느낌을 받을 수 있게끔 구현한 면접실을 3D 가상환경으로 구현하였습니다.

그림 12. 학습실, 면접실, 퀴즈존으로 이동할 수 있는 메인로비

프로그램으로서의 유니티가 아닌 웹에서 제공하는 서비스의 일부로 활용해야 하기 때문에, 3D 그래픽인 Unity 를 2D 환경인 react Web 의 canvas 에 담아야 하였고, WebGL 을 활용하여 구현하였습니다.

11. 확장성을 고려한 Unity 설계

유니티를 단일 씬이 아닌 여러 씬이 유기적으로 연결되게 작성하여 방꾸미기나 여러 씬을 추가할 수 있게 확장성을 고려한 설계를 구현했습니다. 싱글톤 패턴으로 관리자 역할 오브젝트를 만들어서 Network Manager 를 통해서 각각의 씬은 네트워크 상황을 공유하고, Game Manager 을 통해서 환경변수를 포함한 변수와 설정을 공유하게 설정하였습니다.

또한, 플레이어와 경, NPC, 벽과 문 등 다양한 복합 객체에도 Prefab 을 적극적으로 활용하여, 컴포넌트의 재사용성을 높였습니다.

12. Photon Unity Network

유니티에서 타 유저와 상호작용하거나 의사소통할 수 있게 하여, 공감대를 형성하기 쉬운 환경을 제공하였습니다. 그런 환경을 제공하기 위해서 유니티 네트워크에서 사용하는 대중적인 네트워크 솔루션인 PUN2.0 를 사용하여 매치매이킹부터 로비까지 일련의 흐름을 제어하고 관리하였으며, 타인의 방이나 커뮤니티에서 채팅이나 애니메이션과 같은 상호작용을 할 수 있게 설정하였습니다.

그림 13. 각종 상호작용과 사용 설명이 적힌 안내 가이드

Photon Network 는 CDN 을 통해 레이턴시를 줄이고, 가까운 네트워크 환경을 구축하는 것을 도와주지만 안타깝게도 Free 환경에서는 제한이 존재하고, 전용 클라우드 서버를 제공하지 않아 방장의 로컬서버를 사용해야 함으로 인원수에 제한이 있어 대규모의 인원을 수용할 수 있는 커뮤니티를 구현하지는 못하였습니다. (docs 권장 인원 16 명)

13. 게이미피케이션

C;SAFY 서비스를 이용자들의 흥미 및 관심을 가져오고, 서비스 이용에 대한 동기 부여를 주기 위해 뱃지를 이용하여 게이미피케이션을 구현하였습니다.
각 뱃지는 1, 2, 3 레벨 혹은 금 은 동으로 구성하여 사용자의 더 많은 참여를 유도하였습니다.

각 뱃지는 대부분의 서비스와 연동하였고, 별도의 DB 에서 뱃지 데이터를 관리하여 추가/삭제 및 Front-End 에서 뱃지 이름 등의 정보 접근성을 확보하였습니다.
실시간 Badge 획득 구현을 위해 각 획득 시점마다 체크해 주었으며, 그 상황에서 나올 수 있는 Null Pointer Exception 등의 오류들을 트러블 슈팅 하였습니다.

이름	설명	제작 의도
출석 일 수	출석 일 수	사용자의 꾸준한 서비스 이용 유도
등업 뱃지	학습 포인트에 따른 등업 시스템	사용자의 성취욕을 자극하여 학습량 증가 유도
과목 등급	6 가지 과목별 등급	모든 과목의 뱃지를 획득 유도하여 학습량 증가
인강 시청 횟수	인강을 시청한 횟수에 따른 뱃지 획득	인강을 많이 시청하게 하여 학습량 증가 유도
면접 횟수	면접 횟수에 따른 뱃지 획득	면접 경험을 늘리고자 제작
모의고사 풀은 횟수	모의고사 풀은 횟수에 따른 뱃지 획득	모의고사를 여러 번 치러 문제를 자주 접하여 익숙하게 만들기 위함
OX 퀴즈 풀은 횟수	OX 퀴즈를 풀은 횟수에 따른 뱃지 획득	OX 퀴즈를 여러 번 치러 문제를 자주 접하게 하기 위함
4 지선다 문제 풀은 횟수	4 지선다 문제를 풀은 횟수에 따른 뱃지 획득	4 지선다 문제를 여러 번 치러 문제를 자주 접하게 하기 위함
앱 접속 기록	앱 접속 기록이 있으면 획득하는 뱃지	다양한 플랫폼으로 서비스를 이용하게끔 유도

프리미엄 회원	프리미엄 결제를 하면 획득하는 뱃지	프리미엄 사용자 결제 유도
OX 퀴즈 서바이벌 우승	메타버스에서 OX 퀴즈 서바이벌에서 1 등을 하면 획득하는 뱃지	메타버스에서 즐기는 학습 경험을 체험해보도록 유도

- 테이블 뱃지 테이블

게이미피케이션 요소를 메타버스 환경에서도 적용하였습니다. 메인로비에서 카운터로 가면 캐릭터를 따라오는 귀여운 펫을 선택할 수 있습니다. 다양한 펫이 있지만 기본으로 사용할 수 있는 펫은 3 가지입니다. 나머지 펫을 이용하기 위해서는 특정 조건을 만족시켜야 합니다. 사용자의 수집욕을 자극함과 동시에 학습의 효과를 볼 수 있도록 장치를 마련하였습니다.

그림. 펫 선택 UI, 아직 해금되지 못한 펫은 조건이 적혀있다.

그림. 펫을 선택하면 해당 펫이 캐릭터를 따라다닌다.

14. 챗봇과 채팅 WebSocket (+REDIS template)

사용자 간 영상을 동시 시청하고 채팅하는 플레이룸에서 안전하고 빠르게 서비스 제공을 하기 위해 WebSocket STOMP 에 채팅을 관리하는 Redis Template 을 적용하였습니다. Websocket 을 이용하여 TCP 위에서 메시지 스트리밍이 가능하게 설계하였습니다.

또한, STOMP 를 이용하여 메시징 전송의 효율성을 높였습니다. 클라이언트와 서버가 전송할 메시지의 유형, 형식, 내용들을 정의하고, subscriber, sender, broker 를 따로 두어 처리하였습니다. 또한 STOMP 를 이용함으로써 채팅방 여러 개를 동시에 운용할 수 있도록 설계하였습니다.

채팅에는 인 메모리 캐싱을 사용할 수 있는 Redis 를 이용하여 채팅/메시징에 빠른 응답시간을 제공하고, 부하 분산이 가능하도록 만들었습니다.
Redis 는 기존의 DB 이용 보다 36% 좋은 성능, 빠른 응답을 보여주었습니다.

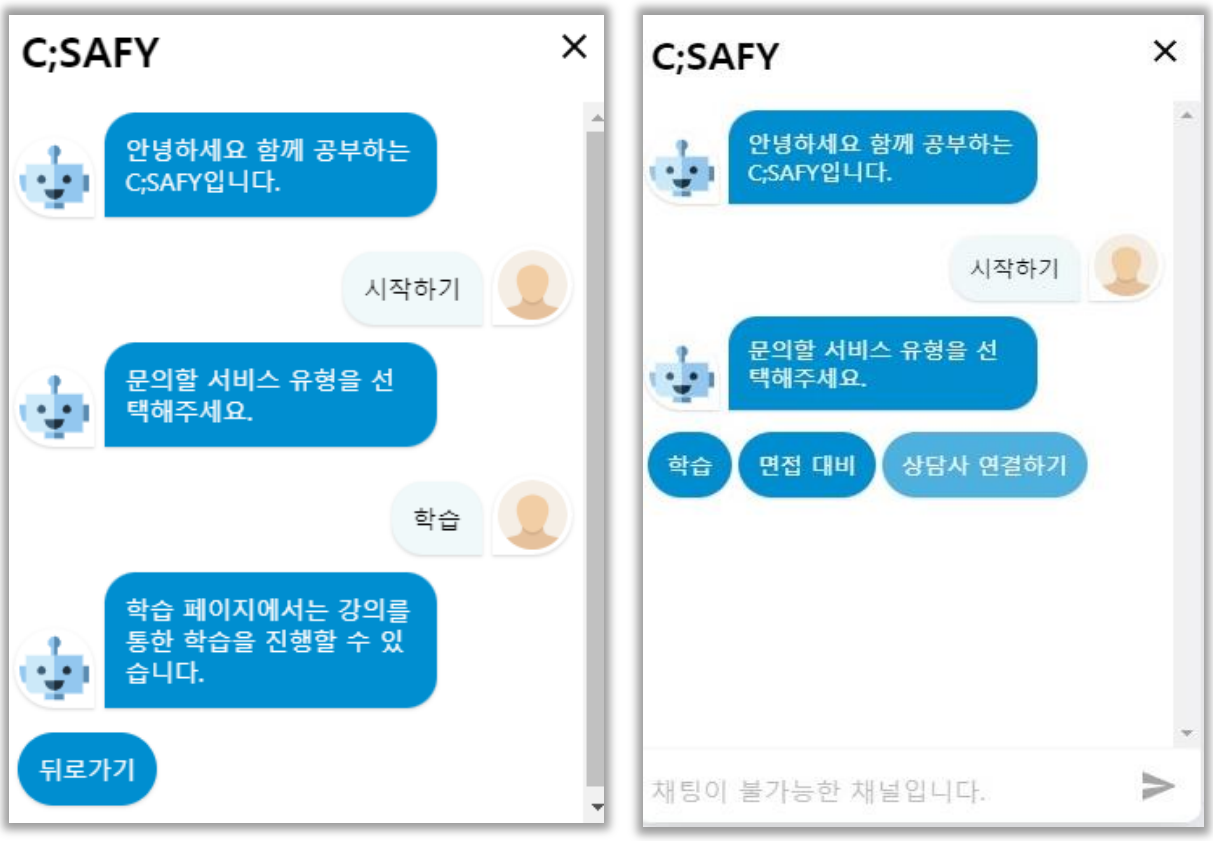
Label	# Samples	Average	Min	Max	Std. Dev.	Error %
메시지 전송	10000	30	24	246	3.17	0.00%
TOTAL	10000	30	24	246	3.17	0.00%

- 그림 채팅/메시징 응답 테이블 (MariaDB)

Label	# Samples	Average	Min	Max	Std. Dev.	Error %
메시지 전송	10000	22	12	42	5.28	0.00%
TOTAL	10000	22	12	42	5.28	0.00%

- 그림 채팅/메시징 응답 테이블 (MariaDB)

[출처 : https://github.com/hwangsero/ChattingProject]



- 그림 챗봇 사용 예시

- 그림 챗봇 상담사 연결 (채팅)

15. EFK + MetricBeats

프로젝트가 MSA 형태로 구성되기 때문에, 흩어져 있는 로그들을 한 곳에 모아 분석하기 위해 EFK 를 통해 로깅 중앙화를 구현하였습니다. 이러한 모니터링 시스템 구축을 통해 원하고자 하는 유의미한 데이터들을 뽑아내고, 시각화 하고자 했습니다.

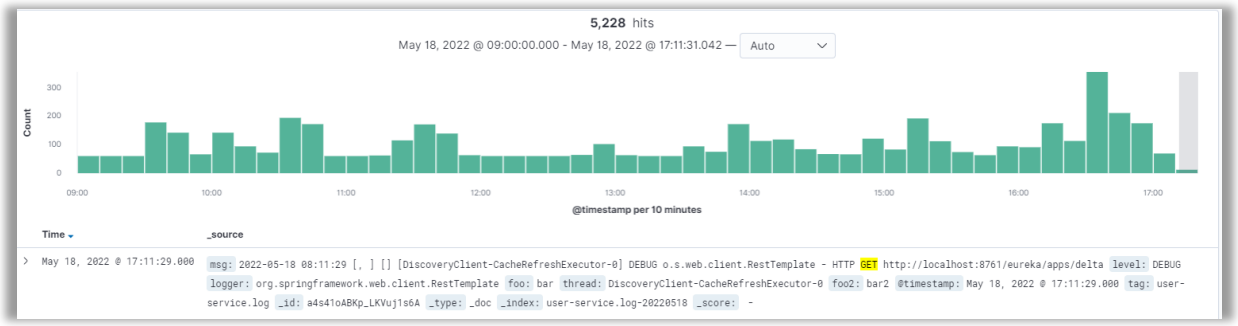
Fluentd 를 데이터 수집기로 사용함으로써 로그 뿐만 아니라 HTTP 혹은 TCP 로부터 전달된 다양한 데이터들을 수집하였고, Fail-Over 를 위해 고가용성 구성을 완료하였습니다. Logstash 를 채택하지 않고 Fluentd 를 채택한 이유는 CRuby 기반으로 자바 런타임이 필요 없고, 향후에 적용할 Kubernetes 와도 더 좋은 성능이나 결과를 얻어낼 수 있기 때문입니다. 또한, logstash 에 비해 더 경량화 된 수집기이므로, 서버에 대한 부담도 덜어 낼 수 있습니다.

ElasticSearch 를 사용하여 Fluentd 에서 모은 데이터, 로그들을 인덱스를 통해 저장합니다. 많은 양의 데이터를 보관할 수 있고, 실시간으로 저장 및 분석, 검색이 가능하도록 해줍니다. 이 검색엔진을 채택한 이유는 빅데이터 처리에 유리하고, 쿼리 속도가 빠르며 확장성이 뛰어나고, 에러에 대한 높은 탄성을 가지기 때문입니다. 또한 kibana 를 제작한 곳과 같은 곳에서 만들어 뛰어난 연계성을 가지기 때문에, 채택하였습니다.

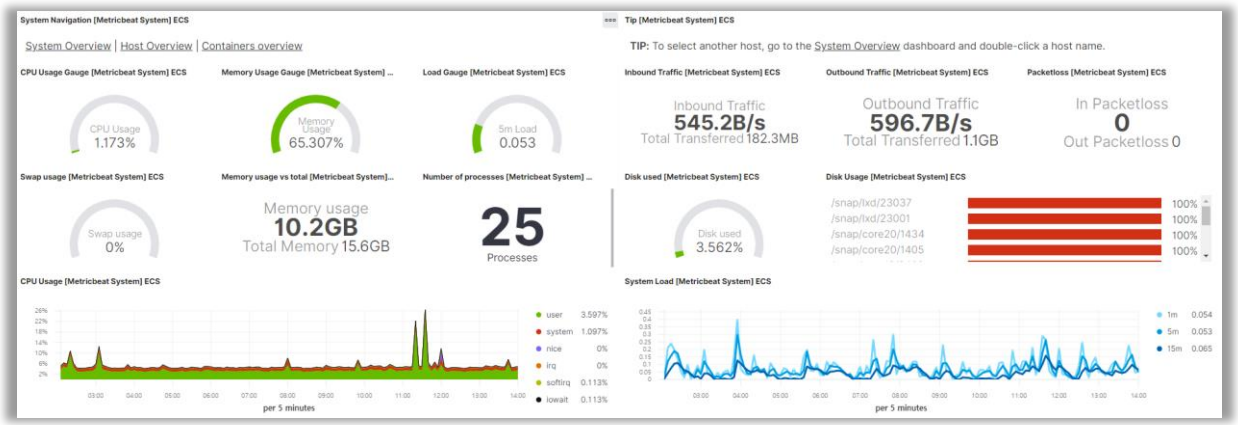
MetricBeats 를 사용하여 각 서비스들을 운용하고 있는 서버의 상태(CPU, RAM, Response Time 등)데이터들을 ElasticSearch 에 전달하였습니다.

마지막으로, Kibana 를 통해 ElasticSearch 에 모아놨던 데이터들에서 유의미한 통계값을 꺼내 대시보드에 시각화하였습니다.

- 그림 Kibana 서비스 로그 및 상태 시각화



- 그림 서버 상태 시각화



- 그림 로그수집

16. Swift

Coordinator패턴을 사용해 View Controller에서 Navigation의 책임을 다른 클래스로 분리했습니다. 흩어져있는 코드를 파악하고 관리하기 쉽게 만들었고, 객체지향적인 프로그래밍을 구현했습니다. 프로토콜 지향 프로그래밍을 사용해 재사용성을 높였고, 디커플링을 통해 유연성을 높였습니다. URL 세션을 이용한 네트워킹, JSON Serializer를 활용했습니다.

17.데이터 수집 및 가공

집중 학습 페이지의 키워드 학습과 4 지선다 문제, OX 퀴즈 문제에 사용될 데이터를 모으기 위해 구글 검색을 통하여 면접 후기와 같은 글을 찾아보았습니다. 특히 개발자라면 준비해야할 Frontend / Backend 기술면접 대비 질문 모음과 같은 글에서 많이 참고하였습니다.

데이터를 모아둔 뒤 분류작업을 하였습니다. 과목별로 자료구조, 컴퓨터구조, 운영체제, 네트워크, 데이터베이스, 그리고 기타로 구성하였습니다. 기타 과목에는 언어 문법이나, 라이브러리 및 프레임워크에 대한 내용, 그리고 최신 트렌드 관련한 내용이 주로 담겼습니다. 분류작업이 끝난 뒤 한 번 더 소분류를 나누었습니다. 그 이유는 기타 과목에 들어간 내용이 너무 두루뭉실 하기도 했고, 더 자세히 분류작업을 해놓으면 이후에 편리하게 사용될 가능성도 생각했기 때문입니다.

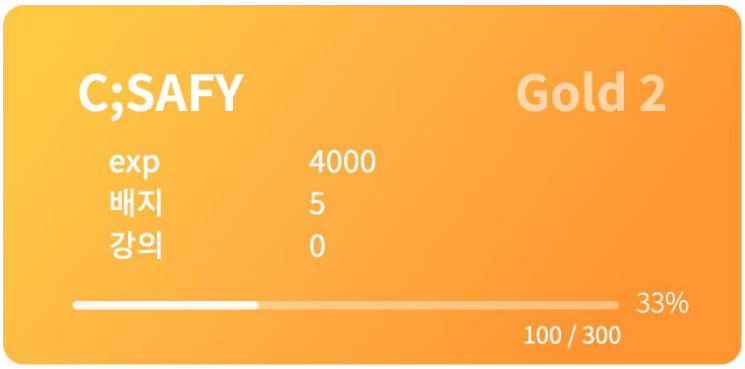
데이터는 질문과 답변으로 구성된 세트가 있었고, 키워드와 그에 대한 설명으로 구성된 세트가 있었습니다. 답변이 충분히 길고 내용을 나누어 4 지선다로 만들 수 있을법한 내용은 4 지선다로 재구성 하였습니다. 각 내용에 대해 답변이 60 자 미만으로 아주 짧은 경우에는 키워드 학습에 사용될 수 있도록 키워드와 답변을 Key 와 Value 로 엮었습니다. 답변이 너무 짧지도, 충분히 길지도 않아서 4 지선다로도, 키워드학습으로도 사용할 수 없는 내용은 '자투리' 라는 이름으로 따로 질문과 답변을 엮었습니다.

4 지선다로 사용할 데이터셋과, '자투리'에 저장된 데이터는 4 지선다 문제와 OX 퀴즈에 이용될 것이므로 정답과 오답 세트가 필요했습니다. 따라서 4 지선다 데이터에 대해서는 각 선택지마다 오답으로 사용될 틀린 값을 작성해주었습니다. 질문을 '~에 대한 설명으로 틀린 것은?' 으로 통일한 뒤, 하나의 오답만 섞여서 전달될 수 있게끔 유도한 방법이었습니다.

키워드 학습에서 사용될 데이터는 Key 로 정해놓은 키워드와, Value 로 정해놓은 답변을 한 쌍으로 전달하여 작동하게끔 하였습니다.

18. Flask 로 카드 꾸미기

별도의 Flask 서브 서버를 구축하여, REST API 로 svg 형태의 반환을 하게하였습니다. Sqlalchemy 를 사용하여, 직접 MYSQL DB 와 연결하여 필요한 정보를 Query 로 Filter 하였고, 경험치 등으로 티어를 계산하여, 카드의 색상이나 경험치 바 등의 지표로 사용하였습니다.



- 그림. 카드 SVG 반환

이후 해당 클라이언트의 요청 content type ‘image/svg+xml’로 반환함으로써 이쪽에서 그려둔 css 를 cdata 형태로 보내줄 수 있었습니다.

해당 방식으로 반환할 경우 마크다운에 이미지를 그릴 수 있고, 이러한 성질을 이용하여 깃허브 등에서 사용할 수 있는 카드로 활용하였습니다.