프로젝트 명세서

기술 스택 및 프로젝트 디자인:

백엔드 by SpringBoot

라이브러리 : netflix-eureka-client, spring-cloud-config, spring-cloud-gateway, lombok, jpa, mysql, mongodb, redis, p6spy, spring-kafka, swagger, gson, aws-s3, komoran, websocket, kakao Oauth2

기술스택

JPA(Java Persistence API) : 자바 ORM(Object Relation Mapping) 기술에 대한 표준 명세로, JAVA에서 제공하는 API임. 자바 어플리케이션에서 관계형 데이터베이스를 사용하는 방식을 정의한 인터페이스임. 우선 ORM에 대해서 알아야 하는데, ORM은 객체와 DB의 테이블을 매핑시켜주는 것을 말함. ORM을 사용함으로써, SQL QUERY로써, DB의 데이터를 다루는게 아닌 JAVA형식으로 메서드를 통해 다룰 수 있음. JPA는 결국 JAVA 내에서 ORM을 사용하기 위한 인터페이스 뭉치에 불과함. JPA := ORM 이라고 생각하면 될 듯함. 해당 기술을 사용해서 프로젝트 내에서 사용할 DB를 컨트롤할 예정임. 연관된 라이브러리로 Data JPA가 있는데, JPA를 쉽게 사용하기 위해서 스프링에서 제공하고 있는 라이브러리임. JPA 자체는 스프링과 관련없는 JAVA 기술 명세인데, Spring에서 더 쉽게 사용할 수 있도록 도와주는 것임.

MongoDB : MongoDB는 오픈 소스 기반 비관계형 데이터베이스 관리 시스템으로 테이블 및 행 대신 유연한 문서(Document)를 활용해 다양한 데이터 형식을 처리하고 저장함. NoSQL 솔루션인 MongoDB는 관계형 데이터베이스 관리 시스템(RDBMS)을 필요로 하지 않으므로, 사용자가 다변량 데이터 유형을 손쉽게 저장하고 쿼리할 수 있는 탄력적인 데이터 저장 모델을 제공함. 이는 개발자의 데이터베이스 관리를 간소화할 뿐 아니라, 뛰어난 확장성을 갖춘 크로스 플랫폼 애플리케이션 및 서비스 환경을 구축함.

Spring Cloud : Spring 란 MSA의 개발, 빌드, 배포, 운영에 필요한 아키텍처를 쉽게 구성할 수 있게 도와주는 SpringBoot 기반 프레임워크임. Spring Cloud를 이용하면 MSA를 위한 환경설정, 서비스 검색, 라우팅, 프록시 등 분산 시스템을 빠르게 설정할 수 있음.

Redis : Redis는 Remote Dictionary Server의 약자로써, 키-값 구조의 비정형 데이터를 저장하고 관리하기 위한 오픈소스 기반의 비관계형 데이터베이스 관리 시스템임. Redis는 인메모리 기반 데이터베이스이므로 쿼리 안정성 향상을 위한 캐시 서버로 도입할 수 있음.

Kafka : Apache Kafka는 분산 스트리밍 플랫폼이며 데이터 파이프라인을 만들 때 주로 사용되는 오픈소스 솔루션임. 카프카는 대용량의 실시간 로그처리에 특화되어 있는 솔루션이며 데이터를 유실없이 안전하게 전달하는 것이 주목적인 메시지 시스템에서 Fault-Tolerant 한 안정적인 아키텍처와 빠른 퍼포먼스를 처리할 수 있음.

Komoran : Komoran은 Shineware에서 자바로 개발한 한국어 형태소 분석기임.

Oauth2 : OAuth2(Open Authorization, Open Authentication 2)는 인증을 위한 표준 프로토콜임. 이 프로토콜에서는 Third-Party 프로그램에게 리소스 소유자를 대신하여 리소스 서버에서 제공하는 자원에 대한 접근 권한을 위임하는 방식을 제공함.

프로젝트 구현 계획

Kafka Messaging queue 를 활용한 Notify Service 동작(Event Driven Microservice) : 현재 프로젝트의 Notify Service에서 생성하는 알림의 종류는 팔로우 알림, 팔로우한 유저의 일지 작성 알림, 일지 좋아요 알림이 있음. 팔로우 기능은 User Service의 비즈니스 로직이며, 일지 작성 및 일지 좋아요 기능은 Plant Service의 기능임. Notify Service가 아닌 다른 Service의 비즈니스 로직들이므로 FeignClient를 통해 API 요청을 보내는 방법도 있지만, 이러한 방법을 사용하여 설계한다면, Service들이 많아질수록 Service 간 의존성 문제 때문에 프로젝트의 복잡성이 증가하게 됨. 이를 해결하기 위해 Event Driven Microservice를 계획한 것임. Service들이 중앙에 있는 Kafka 토픽에 요청을 보내 각각 필요한 요청에 응답하는 식으로 설계하게 된다면, Service들 간의 의존성이 단일 Kafka에 대한 의존성으로 대체되어 복잡성이 사라지며, 동기적 처리 방식인 REST를 Event를 사용한 비동기 처리 방식으로 대신하여 요청 처리속도 또한 높일 수 있음. 이러한 설계 방식을 따라 Notify Service가 User Service, Plant Service가 Kafka topic에 저장한 데이터를 읽어 각각의 요청을 처리할 예정임.

CQRS 패턴과 MongoDB를 통한 데이터 스냅샷 저장 : CQRS 패턴을 구현하기 위해 쓰기 데이터를 저장할 MYSQL과 읽기 데이터를 저장할 MongoDB를 사용할 예정임. 읽기 데이터로 MongoDB를 선택한 이유는 MongoDB는 RDB와는 다르게 쿼리에 대해 최적화된 고유한 데이터 스키마를 사용할 수 있기 때문임. 예를 들어, RDB로 설계할 경우 하나의 태그명에 대한 레코드는 단일한 유저명만 가질 수 있지만, 하나의 Document 데이터는 태그명 당 여러 유저명이 들어가 있는 리스트를 가질 수 있음. 따라서 상당한 쿼리 최적화가 가능해짐. 다만, 쓰기 데이터 DB와 읽기 데이터 DB를 분리하기 위해서는 읽기 데이터 DB가 쓰기 데이터 DB에 동기화되어야 하는데, 이러한 동기화를 위해 Kafka 를 Messaging Queue 로 사용함. Kafka의 Source Connector를 통해 MYSQL에 저장된 값들을 비동기적으로 Topic에 적재한뒤, 해당 Topic의 레코드들로 읽기 데이터 DB를 적재해 나가는 것임. Topic의 레코드들을 읽기 데이터 DB에 적재하는 방법으로는 Kafka Sink Connector와 Listener 애플리케이션 구현이 있음. Sink Connector방식은 Connector만 생성해주면 되지만, MYSQL에 있는 데이터 스키마를 그대로 MongoDB로 가져오기 때문에 쿼리에 최적화된 데이터 스키마

를 따로 만들어줄 수 없음. Listener 애플리케이션 구현 방식은 MYSQL에 레코드가 들어올 때마다, 해당 데이터를 원하는 대로 튜닝하여 MongoDB에 전달할 수 있음. 하지만, 튜닝하는 부분을 개발자가 직접 구현해줘야 함. 현재 프로젝트는 쿼리 최적화를 위해 Listener 애플리케이션 방식을 채택하고 읽기 데이터 DB 설계방식으로 특정 View의 스냅샷 형태를 저장할 예정임. 예를 들어 홈화면에서 필요한 데이터들을 하나의 도큐먼트(레코드)에 모두 저장해 놓는 것임. 따라서 홈화면에 접근할 때마다 테이블에서 원하는 값들을 Join해서 가져올 필요 없이 한번의 Full Scan으로 해결할 수 있음. 다만, 이렇게 스냅샷 형태로 데이터를 저장한다면 Listener를 통해 스냅샷에 있는 데이터가 변경될 때마다 감지하여 갱신해줘야 함. 하나의 도큐먼트 내에 있는 스냅샷 데이터가 변경에 취약한 점 때문에 Listener 애플리케이션 구현의 난이도가 급격히 높아짐.

Redis를 활용한 Batch 처리 : 유저의 좋아요, 팔로우, 조회수와 같이 실시간 트래픽이 높은 데이터는 트랜잭션 부하가 너무 높아 DB에 안좋은 영향을 끼치게 됨. 이를 해결하기 위해 Redis에 실시간 트래픽이 높은 데이터들을 카운팅한 뒤, 스케쥴링을 통해 카운팅된 데이터들을 한번의 트랜잭션으로 해결할 수 있도록 해줌. 이를 Batch 처리 과정으로 볼 수 있음.

Komoran을 활용한 태그 형태소 분석 처리 : 유저가 글을 작성할 때 태그를 사용하여 특정 주제를 강조할 수 있으며, 태그 검색을 통해 해당 글이 조회될 수 있도록 유도할 수 있음. 이러한 태그가 접두사/접미사, 동사 등을 포함한 문장 형태로 되어 있다면, 태그 검색이 사실상 불가능에 가까워지게 됨. 따라서 문장 형태의 태그에 대해 형태소 분석을 사용하여 명사를 뽑아내 태그로 저장해줘야 태그 검색이 수월하게 이루어질 수 있게 됨. 이를 위해 Komoran 라이브러리를 사용하여 형태소 분석을 실행할 계획임.

실시간 채팅을 위한 WebSocket 활용 : 유저 간 실시간 채팅은 Connection이 지속적으로 연결되어 있어야 한다는 특징 때문에 한번의 통신 이후 Connection이 끊어지는 HTTP 프로토콜로는 구현이 불가능함. 따라서 기존의 단방향 HTTP 프로토콜과 호환되며 지속적인 Connection 유지를 제공해주는 WebSocket 프로토콜을 사용하는 게 필수적임. 그러나 모바일 크롬 브라우저와 IE에서는 순수한 WebSocket이 동작하지 않는 문제가 있음. 이를 해결하기 위해서는 WebSocket Emulation을 사용해야 하는데, Spring에서는 SockJS를 제공해줘서 해당 기능을 사용해주면 됨.

부족한 점:

CQRS 패턴을 적용하고, MongoDB에 데이터 스냅샷을 저장하기 위해 Listener 애플리케이션을 구현해야 함. 여기서 Listener 애플리케이션은 데이터 스냅샷을 쓰기 데이터 DB의 최신 상태와 동기화 시켜주기 위해 지속적으로 DB의 Table들을 모니터링 해줘야 하며, Listener 메서드 동작시, 읽기 데이터 조작을 해줘야 함. 따라서 Listener 애플리케이션에서 상당한 부하가 발생할 것이므로, Listener를 다수의 Worker Thread로 동작시켜주는 절차가 필요함.

참조 :

https://www.samsungsds.com/kr/insights/spring_cloud.html

https://engkimbs.tistory.com/691

https://www.ibm.com/kr-ko/cloud/learn/mongodb

https://dev-gorany.tistory.com/224