|  |
| --- |
|  |



|  |
| --- |
| **TUPLI ~Youtube Playlist~** |
| **시스템 기술서** |
|  |
|  |
|  |
|  |
|  |



**목차**

[I. 개요 2](#_Toc96076005)

[1. 프로젝트 개요 2](#_Toc96076006)

[2. 프로젝트 배경 2](#_Toc96076007)

[3. 프로젝트 핵심 기술 2](#_Toc96076008)

[II. 핵심 기술 소개 4](#_Toc96076009)

[1. Spring Data JPA (+Querydsl) 4](#_Toc96076010)

[2. Spring Security (+OAUTH, JWT) 5](#_Toc96076011)

[3. 뱃지를 활용한 게이미피케이션 7](#_Toc96076012)

[4. 취향 분석 및 공유 서비스 9](#_Toc96076013)

[5. WebSocket (+REDIS template) 10](#_Toc96076014)

[6. 네 종류의 DB 사용 11](#_Toc96076015)

[7. 부하테스트 13](#_Toc96076016)

[8. Post 연산을 이용한 성능 향상 16](#_Toc96076017)

[9. 유사도 계산 17](#_Toc96076018)

# 개요

## 프로젝트 개요

공통 프로젝트는 사용자 경험(UX) 중심의 웹 서비스를 완성하는 것이 목표입니다. 본 기술서에서는 이번 프로젝트에서 그 목표 달성을 위해서 어떤 상황과 조건이 주어졌고, 프로젝트를 진행하면서 이를 어떻게 달성해 나갔으며 무엇을 보고 느꼈는지에 대해 정리해보고자 합니다.

## 프로젝트 배경

콘텐츠의 범람이 일어나는 요즘 시대에서 여러분들은 유튜브를 보며 '이 영상, 나만보기 아깝다!' 또는, '나와 비슷한 취향을 가진 사람은 어떤 채널을 구독중일까?' 라는 생각을 해보신 적 있나요?

TUPLI는 그러한 니즈를 충족시켜주기 위해 탄생한 유튜브 콘텐츠 기반 SNS입니다. TUPLI와 함께라면, 유튜브 컨텐츠 공유를 손쉽게! 할 수 있습니다.

## 프로젝트 핵심 기술

이번 프로젝트를 하면서 사용한 핵심 기술들입니다.

* **Spring Web**

전반적인 Rest Controller 구현 및 작업환경을 구축하였습니다.

* **Spring Security**

Spring Security를 통하여 회원가입을 포함한 회원관리, 비밀번호 암호화, 해쉬 기능, jwt유효 검사 등 필터 및 인터셉터 수행, cors, csrf 이슈 해결 등 보안을 포함한 인증, 권한을 관리하였습니다.

* **JWT**

JSON Web Token을 활용하고 Spring Security와 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.yml설정과 연계하여 국제화 메시지를 구현하였습니다.

* **Spring Data JPA**

Spring에서 제공하는 ORM인 Hibernate를 사용하여 DB에 종속되지 않고, 객체 중심의 개발을 할 수 있었고, Spring Data를 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, Getter나 setter을 많이 활용하는 DTO나 의존성 주입이 필요한 클래스들에게 RequiredArgsConstructor 등의 기능을 제공하여 코드를 간략하게 짤 수 있었습니다.

* **MySQL MariaDB**

관계형 DB인 MariaDB를 메인 DB로 사용하였습니다.

* **AWS (EC2 + RDS)**

EC2 인스턴스에서 서버를 구축하고, RDS 서버를 이용하여 관계형 DB를 적재하였습니다.

* **Google Cloud Storage**

프로필 이미지 등 static한 이미지를 저장하기 위하여 Google Cloud Platform에서 제공하는 Google Cloud Storage를 이용하였습니다.

* **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.

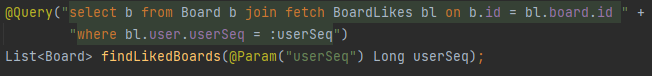
# 핵심 기술 소개

## Spring Data JPA (+Querydsl)

기존의 Mybatis 같은 프레임 워크를 통한 DB중심의 쿼리 질의에서, JPA나 QueryDsl를 이용하여 객체 중심의 개발이 이루어 질 수 있도록 구성하였습니다.

코드로 쿼리를 작성하기 때문에, 컴파일 시점에 오류를 확인할 수 있었습니다. JPA는 기본 메소드를 제공해주어 기본 쿼리에 대해서는 빠른 코딩을 할 수 있도록 도와주었고 QueryDsl은, 동적 쿼리를 이용해 쿼리를 유연하게 짤 수 있게 만들어 더 효율적인 코딩이 가능하도록 만들어 주었습니다. 또한, Pageable을 이용하여 무한 스크롤을 구현하였습니다.

Fetch Join을 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.



- 그림 fetch join을 이용한 N + 1 이슈 해결

텍스트이(가) 표시된 사진

자동 생성된 설명

-그림 hibernate.default\_batch\_fetch\_size를 이용하여 쿼리가 in절로 묶인 모습

텍스트이(가) 표시된 사진

자동 생성된 설명

- 그림 pageable을 통한 페이징, 무한 스크롤 구현

## Spring Security (+OAUTH, JWT)

현 프로젝트가 기존의 구글, 네이버와 같은 보안이 검증된 사이트 보다 보안이 더 우수하다고 볼 수 없기 때문에, 로그인에 OAUTH2를 적용하여 보안성을 올릴 수 있도록 설계하였습니다. 특히, 여러 OAUTH 인증 방식 중에서 Authorization Code Grant 인증 방식을 사용하여 Access Token을 직접적으로 Front-End에 전달하지 않고 Backend에서 Access Token을 검증한 후 JWT Token을 만들어 전달해 주면서, Access Token이 탈취되지 않도록 구성하였습니다.

Jwt Token은 SecretKey를 이용하여 builder를 통해 토큰에 저장하고 싶은 데이터들을 넣고, 토큰 만료시간을 지정한 후 HS256을 이용하여 발급했습니다. HS256 대칭키 암호화 알고리즘을 사용하여 해시 값의 특징을 활용할 수 있게 설계하였고, 이를 통해 메시지 변조여부를 확인할 수 있도록 설계하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

- 그림 JWT Token 생성

Spring Scheduler, BulkSQL : Spring Schedule에 cron을 이용하여 실시간 검색어 트렌드 순위를 정할 점수 선정작업을 구현하였고, 각 작업 마다 실행되는 대용량의 데이터에 대한 update 혹은 delete 쿼리에 벌크 연산을 사용하여 서버 부하, 실행 시간을 줄였습니다.

벌크 연산 사용 시, 컨텍스트 flush 및 clear 작업을 추가해주어 영속성 컨텍스트/2차캐시와 DB의 데이터가 다를 수 있는 문제를 해결할 수 있도록 설계하였습니다.

텍스트, 스크린샷, 모니터, 화면이(가) 표시된 사진

자동 생성된 설명

- 그림 벌크 연산

## 뱃지를 활용한 게이미피케이션

Tupli 서비스를 이용자들의 흥미 및 관심을 가져오고, 서비스 이용에 대한 동기 부여를 주기 위해 뱃지를 이용하여 게이미피케이션을 구현하였습니다.

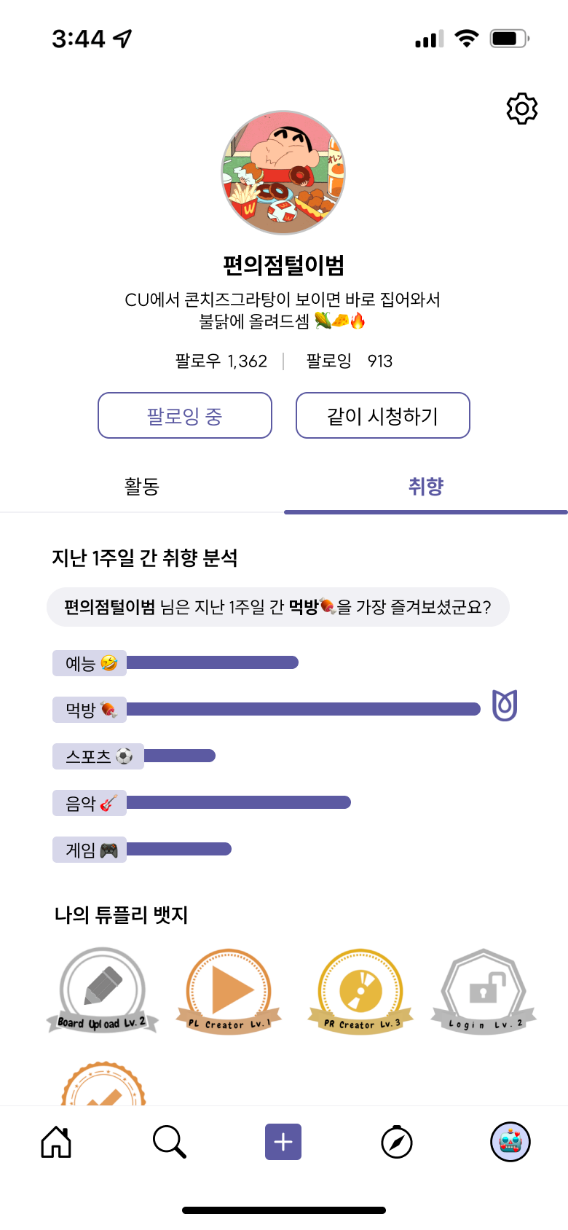
각 뱃지는 1, 2, 3레벨 혹은 금 은 동으로 구성하여 사용자의 더 많은 참여를 유도하였습니다.

각 뱃지는 대부분의 서비스와 연동하였고, 별도의 DB에서 뱃지 데이터를 관리하여 추가/삭제 및 Front-End에서 뱃지 이름 등의 정보 접근성을 확보하였습니다.

실시간 Badge 획득 구현을 위해 각 획득 시점마다 체크해 주었으며, 그 상황에서 나올 수 있는 Null Pointer Exception등의 오류들을 트러블 슈팅 하였습니다.

|  |  |  |
| --- | --- | --- |
| 이름 | 설명 | 제작 의도 |
| 로그인 횟수 | 로그인 횟수 | 사용자의 서비스에 대한 다 회 접근 유도 |
| 출석 일 수 | 출석 일 수 | 사용자의 꾸준한 서비스 이용 유도 |
| 게시글 게시 수 | 게시글 게시 수 | 서비스의 게시판 활성화 |
| 플레이리스트 게시 수 | 플레이리스트 제작, 게시 수 | 플레이리스트 제작 활성화를 통한 많은 플레이룸 제작 유도 |
| 플레이룸 게시 수 | 플레이룸을 만든 횟수 | 사용자들의 플레이룸 이용 빈도 상승 유도 |
| 팔로우 수 | 자신이 팔로우 한 유저 수 | 서비스의 SNS기능 사용 유도 및 취향 데이터 집계 |
| 팔로워 수 | 자신을 팔로우하는 유저 수 | 서비스의 SNS기능 사용 유도 및 취향 데이터 집계 |
| 플레이룸 최고 시청자 수 | 자신의 플레이룸 중 최고 시청자 수 | 사용자들 간의 경쟁을 유도하여 더 많은 서비스 유입 유도 |
| 프리미엄 회원 | Tupli의 프리미엄 회원 여부 | 프리미엄 회원임을 보여주어 다른 유저도 이용하도록 유도 |
| 총 시청시간 | 플레이룸에서 시청한 시간 | 플레이룸 시청 유도 |
| 각 장르별 시청시간 | 플레이룸에서 상영되는 비디오장르 별 시청시간 | 사용자의 취향분석 데이터 수집 및 사용자 간 취향공유 유도 |

- 테이블 뱃지 테이블

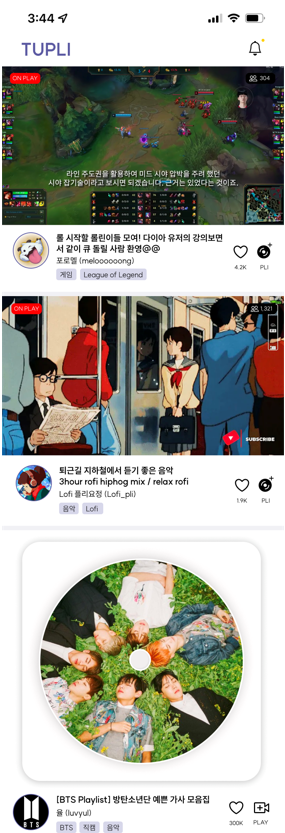


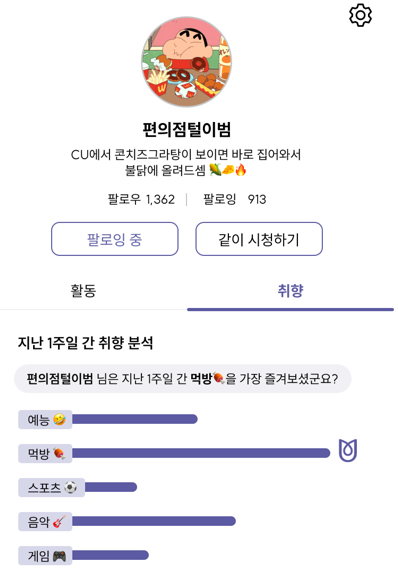
- 그림 사용자 뱃지 및 게이미피케이션이 적용된 프로필

## 취향 분석 및 공유 서비스

서비스 내 추천 시스템을 이용하기 위해 각 사용자 별로 취향을 DB에 저장하였습니다. 또한 사용자 간 취향 공유 서비스를 제공하여, 사용자 간 취향 확인 및 취향 갱신을 할 수 있도록 설계하였습니다.

기본 취향을 입력받은 취향 데이터에 해당 장르의 시청 시간을 접목하고 그를 취향 점수로 만들어 DB에 사용자 별로 저장하여, 해당 사용자의 취향에 맞는 플레이룸, 플레이리스트를 보여주고, 그를 사용자에게 시청할 수 있도록 설계하였습니다.





- 그림 타 사용자 취향 확인, 공유

- 그림 메인페이지 사용자 별 추천

## WebSocket (+REDIS template)

사용자 간 영상을 동시 시청하고 채팅하는 플레이룸에서 안전하고 빠르게 서비스 제공을 하기 위해 WebSocket STOMP 에 채팅을 관리하는 Redis Template을 적용하였습니다.

Websocket을 이용하여 TCP 위에서 메시지 스트리밍이 가능하게 설계하였습니다.

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

채팅에는 인 메모리 캐싱을 사용할 수 있는Redis를 이용하여 채팅/메시징에 빠른 응답시간을 제공하고, 부하 분산이 가능하도록 만들었습니다.

Redis는 기존의 DB이용 보다 36% 좋은 성능, 빠른 응답을 보여주었습니다.

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

텍스트이(가) 표시된 사진

자동 생성된 설명

- 그림 플레이룸 영상 동시시청 및 채팅

## 네 종류의 DB 사용

서비스의 성능, 응답시간을 향상시키고, 부하 경감을 위해 각 기능에 적합한 4가지 DB를 사용하였습니다.

프로젝트의 기본이 되는 메인 DB로 Maria DB를 사용했습니다. 인덱스, 레코드 데이터 모두 메모리 캐시를 이용할 수 있어 MySQL에서 사용하는 MySAM 스토리지 엔진에 비해 빠른 처리를 가질 수 있어 메인 DB로 선택하였습니다.

두 번째는 Redis입니다. Redis는 Websocket과 사용하여 플레이룸 및 채팅에 적용했습니다. 인 메모리 캐싱 방법을 이용하는 DB 이므로, 기존의 oracle DB보다 응답시간을 약 36% 빠르게 가져올 수 있었습니다. 이는 영상과 채팅이 모두에게 빠르게 전달되어야 하는 플레이룸에서 적합하다고 판단하여 적용하였습니다.

세 번째는 GC Storage(Google Cloud Storage)입니다. 서비스에서 빠르게 자주 가져오고, 크기가 작다는 프로필 이미지의 특성상, GC Storage를 이용하였습니다. HDFS와는 다르게 node가 시작되자 마자 작업을 시작할 수 있어, 빠르게 프로필 이미지를 가져올 수 있었습니다. 또한, 권한 관리를 Google IAM을 통해서 적용하여 따로 권한 관리를 할 필요가 없고, global하게 데이터의 일관성을 보장하여, 프로필 이미지 데이터에 문제가 생기지 않도록 설계하였습니다.

마지막으로 RTDB(RealTimeDB)입니다. 서비스에서 바로 값을 바꿔 보여주어야 하는 상황에 적용하기 위해 RTDB를 선택하였습니다. DB에서 값이 바뀔 때 마다 front-end 단으로 곧장 변경이 적용된다는 장점을 이용하여, 실시간으로 바뀌어야 하는 유저 별 알림 정보 등에 적용해 사용자에게 빠른 응답속도, 반응을 보여주도록 설계하였습니다.

텍스트, 명함이(가) 표시된 사진

자동 생성된 설명

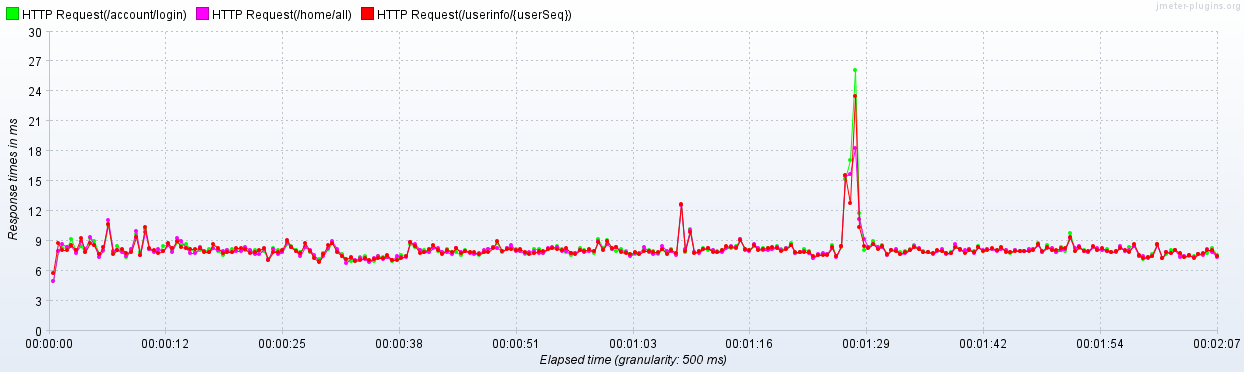
- 그림 프로젝트에 사용한 4가지 DB

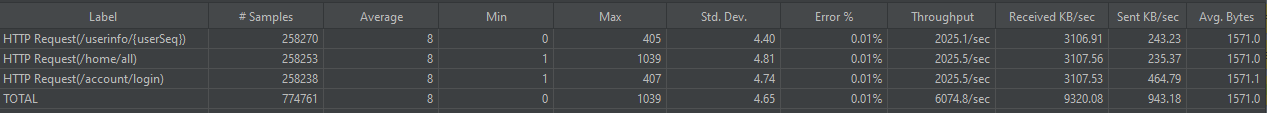
## 부하테스트

Jmeter를 이용하여 사용자가 서비스에서 자주 요청할 Http request들에 부하테스트를 진행하였습니다.

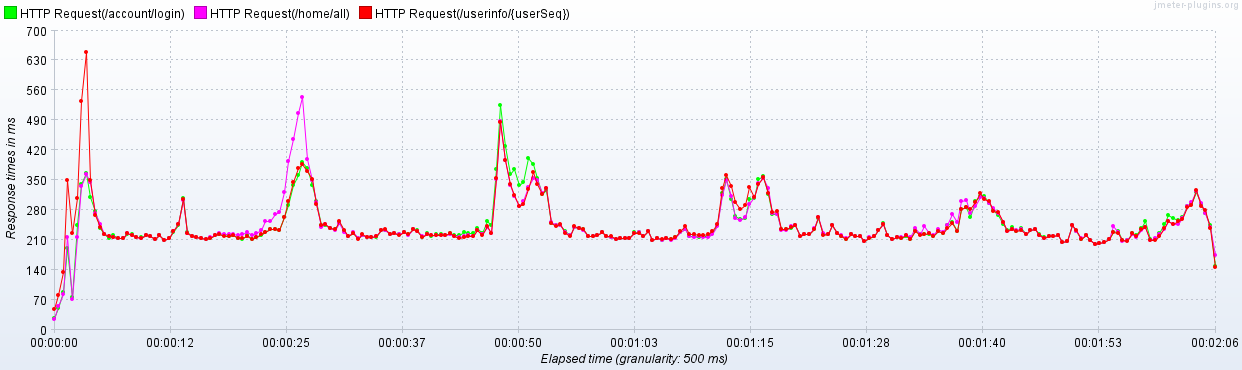
평균 응답속도가 어떻게 나오는 지, 트랜잭션이 얼마나 동작하는 지, 에러율 등을 직접 판단하였습니다. 응답시간이 최초로 0ms의 국자 모양이 나올때의 에러율이 1.15%였기 때문에, 에러율이 1%미만이라면 사용자가 서비스 이용에 문제가 없을 것이라고 판단하여, 이를 기준으로 최대 동시 사용자 수를 결정하였습니다.

테스트에 사용할 request는 로그인, 메인 페이지 접근, 유저 프로필 접근 같이 자주 요청하는 Http request로 결정하였습니다.

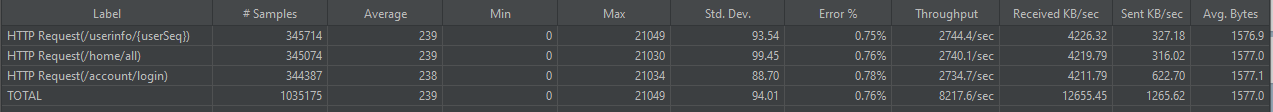
- 그림 50명 동접 응답 시간 그래프



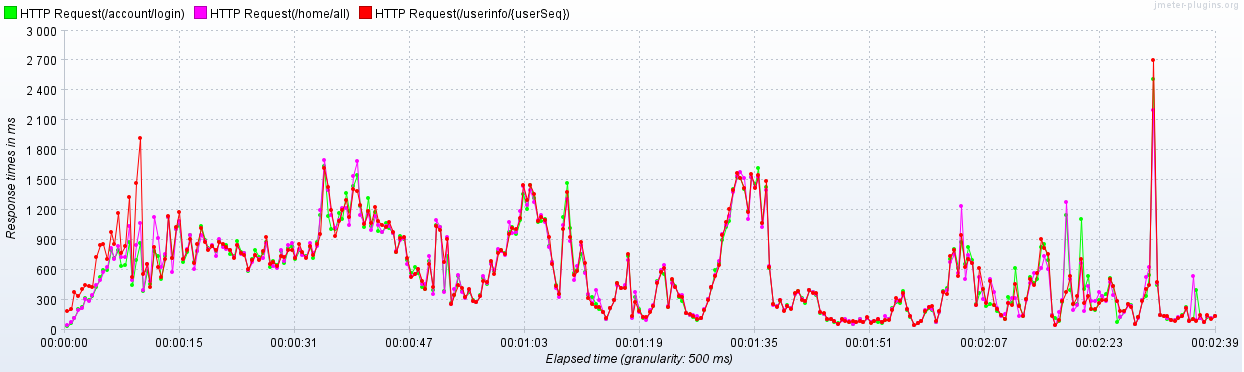
- 그림 50 명 동접 결과 테이블

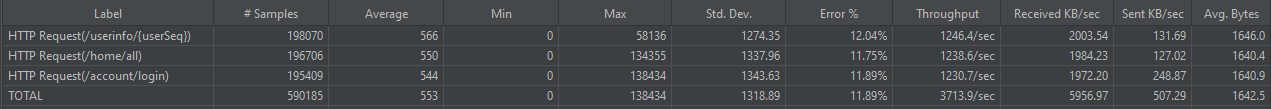


- 그림 2000명 동접 응답시간 그래프 ( 최대 유저 수 )



- 그림 2000명 동접 결과 테이블 ( 최대 유저 수 )

- 그림 4000명 동접 응답시간 그래프 ( 과부하 오류 )



- 그림 4000명 동접 결과 테이블 ( 과부하 오류 )

## Post 연산을 이용한 성능 향상

복잡한 계산을 POST에서 구현하고, GET에서 결과를 가져가는 방식으로 설계하였습니다.

2000년 트위터 API 요청 비율 그림을 보시면, POST요청이 10%미만, GET이 90% 이상의 요청을 부르고 있다는 통계를 나타내고 있습니다.

따라서 플레이리스트에서 비디오를 분리하고 취향 분석 데이터를 뽑아 저장하고, 플레이룸에서 플레이리스트를 분리하고, 비디오를 분리하는 복잡한 연산들을 Post 과정에서 처리하였습니다.



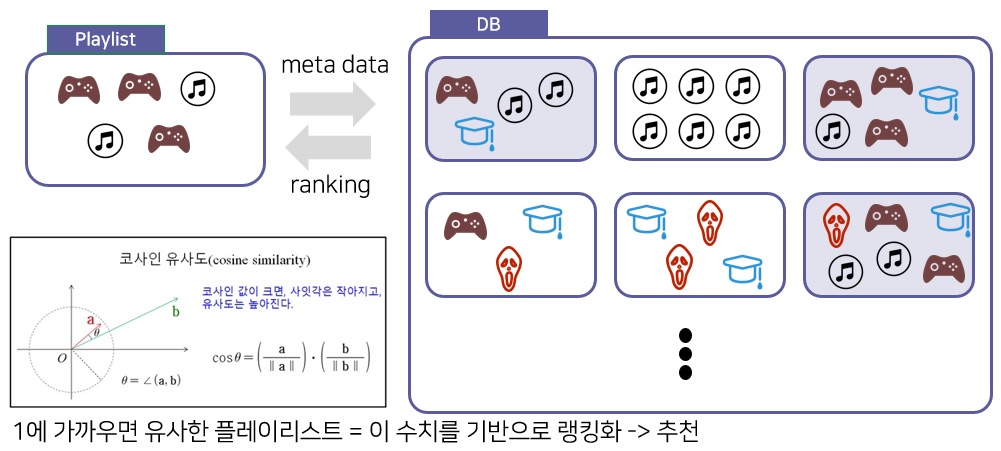
- 그림 트위터 API 요청 비율

## 유사도 계산

영상의 뭉치인 플레이리스트의 메타정보를 태그 뭉치라고 가정한다면

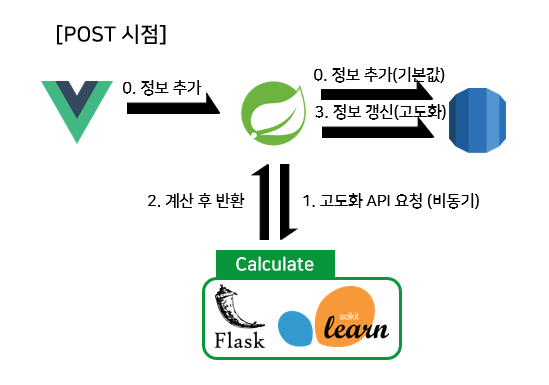
{ 22:1 23:2 24:1 } 라고 표현할 수 있을 것이고, 이 태그 뭉치는 1. { 22:2 23:1 } 2. { 18:1 28:1 } 중에 1과 유사할 것이다

이를 계산하기 위해, 빅데이터 머신러닝 라이브러리인 사이킷 런을 사용하여 Dict Vectorizer를 사용하였다. Vectorizer을 사용하여 해당 메타데이터들을 정보를 가진 행과 열로 표현할 수 있었고, 각 행과 열에 대한 코사인 유사도 계산을 통해 해당 플레이리스트와 유사한 정도를 파악하고 추천할 수 있었다.



- 그림 코사인 유사도 파악한 랭킹화와 추천.

이러한 연산의 경우 대분류, 중분류 후 약 3만건의 플레이리스트가 연산 대상일 때 6초의 연산시간이 필요한데, 실제 서버에서 이러한 연산에 많은 시간을 쓰게 되면 다른 처리를 할 수 없게 된다. 따라서 이 계산 만을 계산하기 위한 서버를 별도로 구축하고, 메인 서버에서 비동기화 요청을 보내는 방식으로 이 문제를 해결하였다.



- 그림 메인서버에서의 비동기화 요청