|  |
| --- |
|  |



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



**목차**

[I. 개요 2](#_Toc96459770)

[1. 프로젝트 개요 2](#_Toc96459771)

[2. 프로젝트 배경 2](#_Toc96459772)

[3. 프로젝트 핵심 기술 2](#_Toc96459773)

[II. 핵심 기술 소개 5](#_Toc96459774)

[1. Spring Data JPA (+Querydsl) 5](#_Toc96459775)

[2. Spring Security (+OAUTH, JWT) 6](#_Toc96459776)

[3. Spring Scheduler, BulkSQL 7](#_Toc96459777)

[4. 뱃지를 활용한 게이미피케이션 7](#_Toc96459778)

[5. 취향 분석 및 공유 서비스 9](#_Toc96459779)

[6. WebSocket (+REDIS template) 11](#_Toc96459780)

[7. 네 종류의 DB 사용 12](#_Toc96459781)

[8. 부하테스트 13](#_Toc96459782)

[9. Post 연산을 이용한 성능 향상 16](#_Toc96459783)

[10. 유사도 계산 16](#_Toc96459784)

[11. 비동기 연산 요청 17](#_Toc96459785)

[12. 플레이룸 영상 싱크 18](#_Toc96459786)

13. Youtube API 22

14. UI/UX 23

15. 태그 추천 32

16. FE 컴포넌트 구성 33

# 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.

* **WebSocket**

웹 상에서 간단히 소켓 통신을 하게 해주는 spring 추천 공식 dependecy입니다. STOMP 메시징 프로토콜과 함께 해당 dependecy를 이용하여 채팅 서버와 영상 동기화의 메시징 처리(Sub(구독) /Pub(발행))를 구현하였습니다.

* **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.

* **Let’s Encrypt**

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

* **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.

* **NoSQL Redis**

‘Key-value’ 구조인 비관계형 DB입니다. 채팅 서버와 영상 동기화의 메시징 처리(Sub(구독) /Pub(발행))를 위해 사용하였습니다.

TTL(Time to Live)기능을 이용한 자동 소멸을 이용해, 특정 정보를 시간 후에 사라지는 기능이나 과거 채팅 로그 기록도 구현해 두었지만, 무료 DB로 운영하기에는 용량 사용량이 커서 비활성화 해두었습니다.

* **Firebase RTDB (Realtime DB)**

‘Key-value’ 구조인 비관계형 DB입니다. 실시간 알림을 브라우저 자체 기능을 이용하지 않고도 구현할 수 있게 하는 서비스를 제공해 주었습니다.

* **Google Cloud Storage**

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

* **AWS (EC2 + RDS)**

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

* **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 서버에서 결제 내역과 내용을 DB에 저장할 수 있게 구축하였습니다.

* **Kakao 링크 공유**

Kakao API를 이용하여 서비스 이용자의 카카오톡 계정과 연계해 쉽고 빠르게 플레이룸 초대 링크 등 서비스를 공유할 수 있게 하였습니다. 비회원에게도 제공되는 접근 장벽이 낮은 서비스에 대해 대다수 사용자에게 익숙한 카카오톡을 이용한 공유를 제공함으로써 서비스 초기 확장 가능성을 넓히기 위해 노력했습니다.

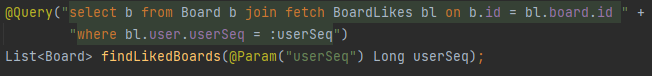
# 핵심 기술 소개

## 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 자식 엔티티를 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 않고 차후 필요하다면 Refresh Token으로 활용할 수 있도록 구성하였습니다.

Jwt Token은 Secret Key를 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 Scheduler에 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.

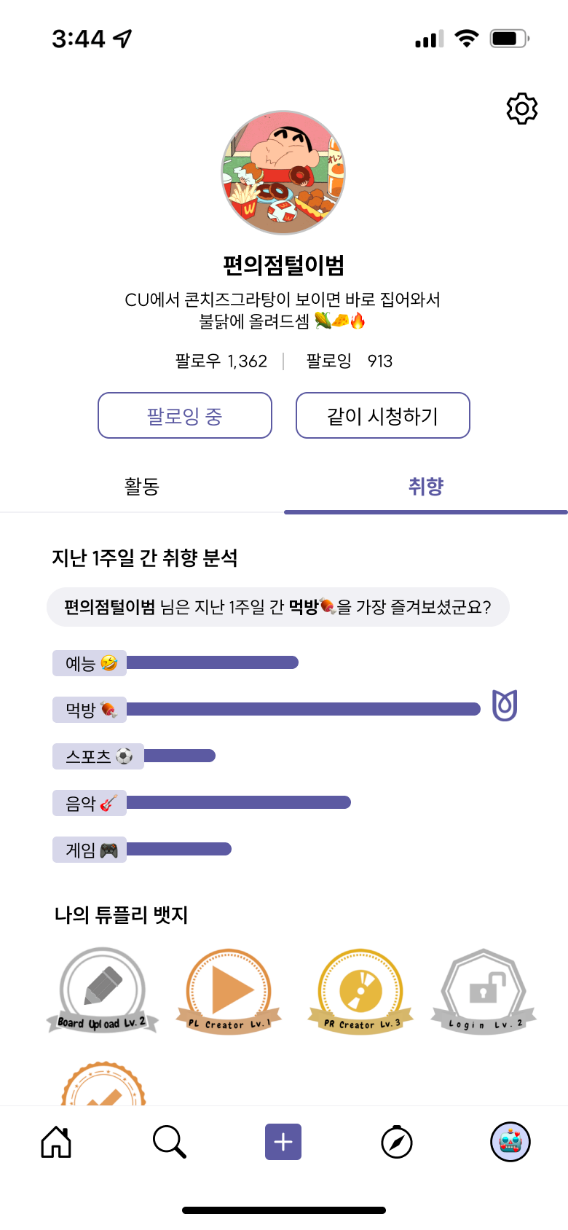
각 뱃지는 1, 2, 3레벨 혹은 금 은 동으로 구성하여 사용자의 더 많은 참여를 유도하였습니다.

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

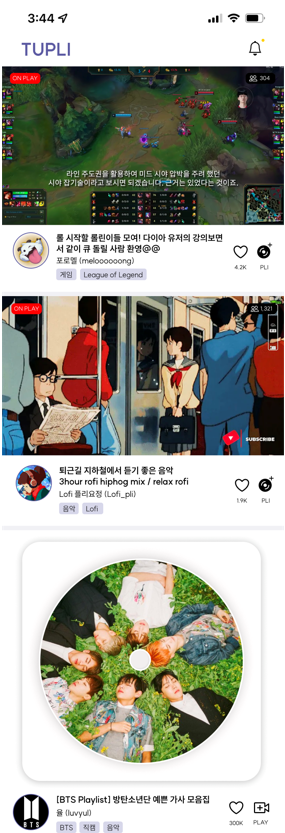
서비스 내 취향 분석 시스템 및 추천 시스템을 이용하기 위해 각 사용자 별로 취향을 DB에 저장하였습니다. 또한 사용자 간 취향 공유 서비스를 제공하고 내용을 확인할 수 있게 구성하여, 사용자 간 취향 확인 및 취향 갱신을 할 수 있도록 설계하였습니다.

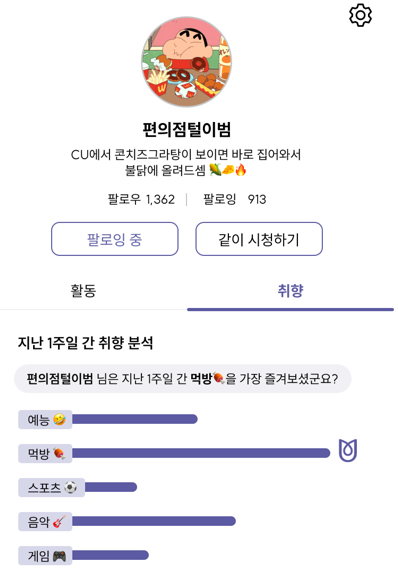
취향 분석에 쓰일 데이터는 유저가 해당 서비스를 이용하며 행하는 행동에 의미와 점수를 부여하여 수집하였습니다.

|  |  |
| --- | --- |
| 이름 | 점수 |
| 플레이리스트 제작 | 소속 영상 카테고리당 3점 |
| 플레이룸 제작 | 소속 영상 카테고리당 2점 |
| 영상에 좋아요 하기 | 해당 영상 카테고리 4점 |
| 영상에 좋아요 취소 하기 | 해당 영상 카데고리 -3점 |
| 플레이리스트 좋아요 하기 | 소속 영상 카테고리당 1점 |
| 플레이룸 좋아요 하기 | 소속 영상 카테고리당 2점 |
| 플레이룸 방문 | 소속 영상 카테고리당 1점 |

- 테이블 취향 분석

해당 장르의 시청 시간을 접목하고 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 depenency를 활용하여 STOMP 프로토콜을 통한 메시징을 구현하였고, 고도화 과정으로 더 빠른 응답속도로 메시징을 전달하고, 동시에 채팅 내역도 관리할 수 있는 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 Database

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

프로젝트의 기본이 되는 메인 DB로 Maria DB를 사용했습니다. Inno\_DB엔진을 사용하여 인덱스, 레코드 데이터 모두 메모리 캐시를 이용할 수 있어 MyISAM 스토리지 엔진에 비해 빠른 처리를 가질 수 있어 해당 스토리지 엔진으로 메인 DB로 선택하였습니다.

두 번째는 Redis입니다. Redis는 Websocket과 사용하여 플레이룸 및 채팅에 적용했습니다. 인 메모리 캐싱 방법을 이용하는 DB 이므로, 기존의 RDB를 사용하는 방식보다 응답시간이 약 36% 빨라졌습니다. 이는 영상 정보와 채팅이 담긴 메시징이 모두에게 빠르게 전달되어야 하는 플레이룸에서 적합하다고 판단하여 사용하였습니다.

세 번째는 GC Storage(Google Cloud Storage)입니다. 서비스 내에서 이미지를 사용하려면 static한 이미지 주소 설정이 필요한데, EC2 내부에서는 이러한 환경을 갖추기가 쉽지 않기 때문에 프로필 이미지 특성에 맞게 이미지의 크기와 용량을 최적화 하여 GC Storage에 저장하는 방법을 사용하였습니다. 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 Firebase RTDB (RealTimeDB)입니다. 서비스에서 바로 값을 바꿔 보여주어야 하는 상황에 적용하기 위해 RTDB를 선택하였습니다. DB에서 값이 바뀔 때 마다 request 없이도 front-end 단으로 곧장 변경이 적용된다는 특징을 이용하여, 실시간으로 바뀌어야 하는 유저 별 알림 정보 등에 적용해 사용자에게 빠른 응답속도, 반응을 보여주도록 설계하였습니다.

텍스트, 명함이(가) 표시된 사진

자동 생성된 설명

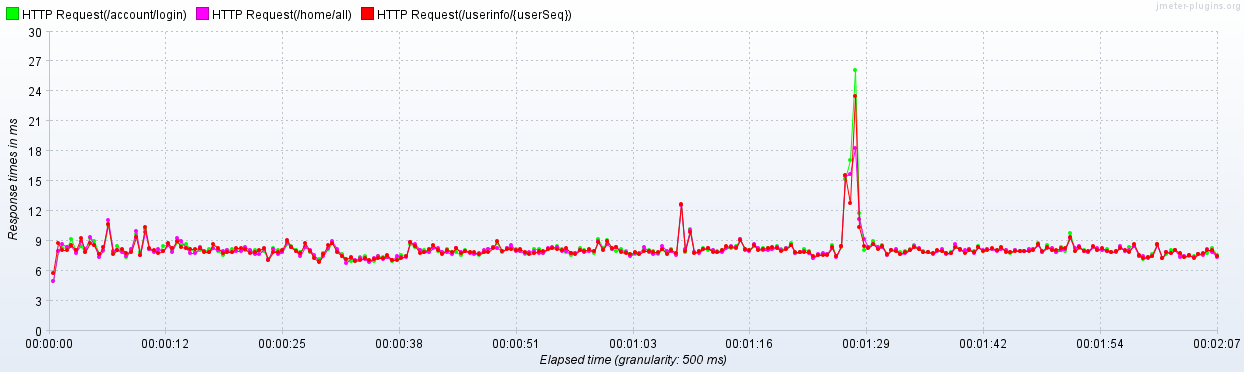
- 그림 프로젝트에 사용한 4가지 DB

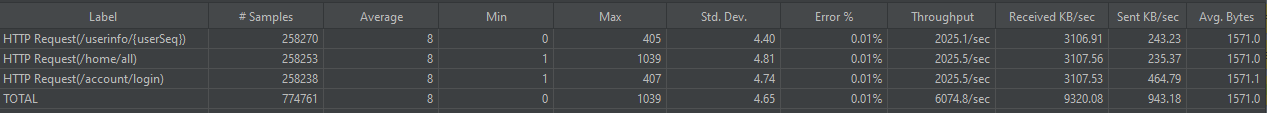
## 부하테스트

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 예상 최대 동시 사용자 수를 결정하였습니다.

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

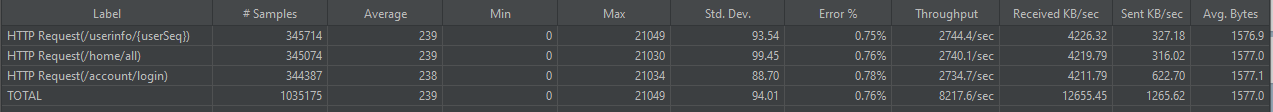
- 그림 50명 동접 응답 시간 그래프



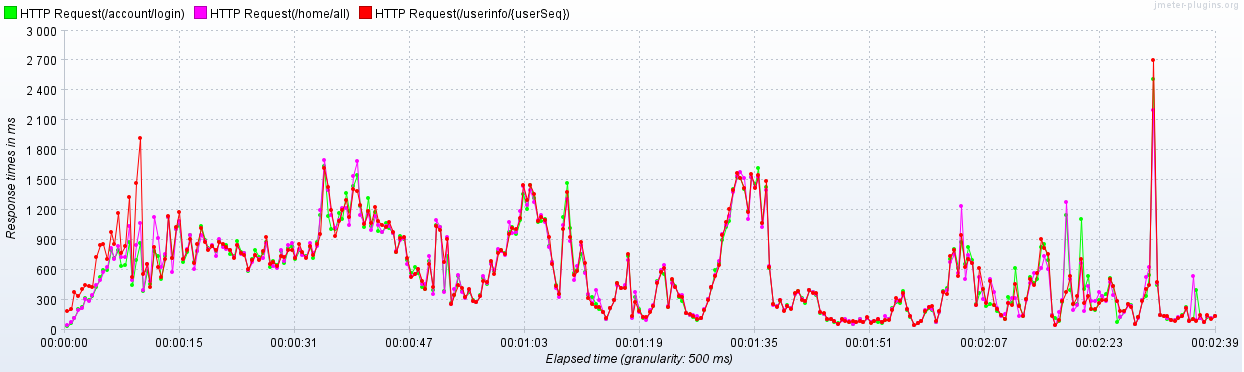
- 그림 50 명 동접 결과 테이블

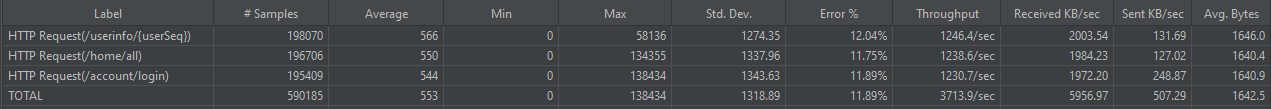


- 그림 2000명 동접 응답시간 그래프 ( 최대 유저 수 )



- 그림 2000명 동접 결과 테이블 ( 최대 유저 수 )

- 그림 4000명 동접 응답시간 그래프 ( 과부하 오류 )



- 그림 4000명 동접 결과 테이블 ( 과부하 오류 )

## Post/Get 시점에 따른 성능 향상



- 그림 트위터 API 요청 비율

2000년 트위터 API 요청 비율 그래프를 보시면, 전체 API 요청 중 POST요청이 10% 미만이고, GET이 90% 이상의 요청을 부르고 있다는 통계를 나타내고 있습니다.

따라서 저희는 복잡한 계산을 POST에서 구현하고, GET에서 결과를 가져가는 방식으로 설계하였습니다.

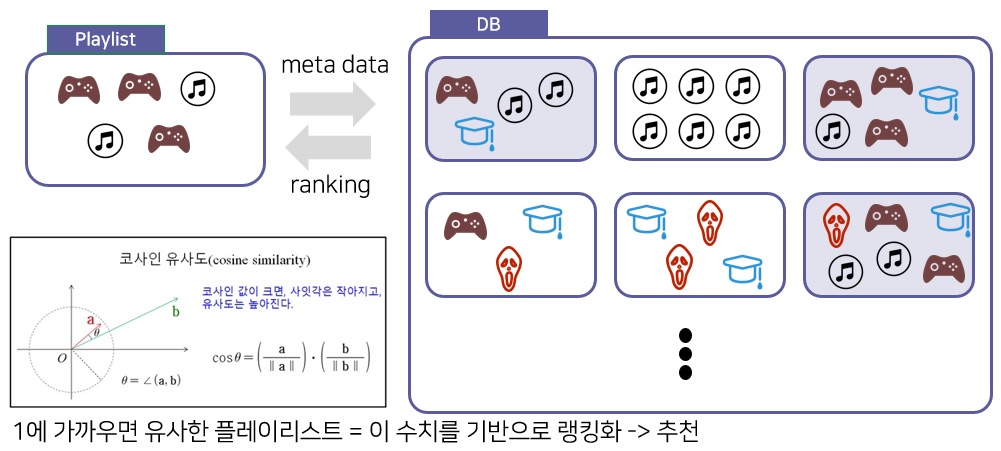
예를 들어 플레이리스트나 플레이룸을 생성하는 POST 과정에서 비디오 분리나 취향 분석을 미리 뽑아 연산해 두어 저장하는 등 복잡한 연산을 미리 해두고, 관련 정보 GET 요청이 들어왔을 때는 계산을 최소화하여, 이미 계산해둔 결과를 보내는 식으로 처리하였습니다.

## 유사도 계산

영상의 뭉치인 플레이리스트의 메타정보를 태그 뭉치라고 가정하여 유사도 측정을 하였습니다.

따라서, 예를 들어 { 22:1 23:2 24:1 } 라는 정보를 가지는 플레이리스트가 있다면, 이 메타 정보는 { 22:2 23:1 }라는 메타 정보를 가진 플레이리스트1과 { 18:1 28:1 }라는 메타정보를 가진 플레이리스트 2 중에서 플레이리스트 1과 유사할 것입니다.

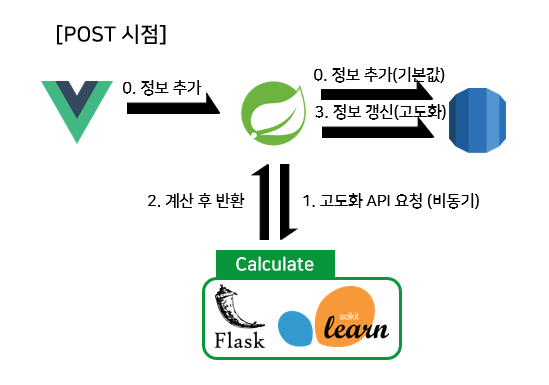
이를 계산하기 위해, 빅데이터 머신러닝 라이브러리인 사이킷 런을 사용하여 Dict Vectorizer를 사용하였습니다. Vectorizer을 사용하여 해당 메타데이터들을 정보를 가진 행과 열로 표현할 수 있었고, 각 행과 열에 대한 코사인 유사도 계산을 통해 해당 플레이리스트와 유사한 정도를 파악하고 추천할 수 있었습니다.



- 그림 코사인 유사도 파악한 랭킹화와 추천.

## 비동기 연산 요청

추천을 위한 백 서버를 운영하는데 있어서, 추천 연산의 경우 대분류, 중분류 후 약 3만건의 플레이리스트가 연산 대상이라고 가정할 때 6초의 연산시간이 필요하다는 것이 확인되었습니다. 실제 서버에서 이러한 연산에 많은 시간을 쓰게 되면 다른 처리를 할 수 없게 되고, 이 정보는 저장할 정보이기 때문에 즉각적인 연산 결과가 필요한 연산이 아닙니다. 따라서 이 계산을 메인 서버에서 추천 계산 서버인 Flask로 비동기로 요청하고, 그 결과를 메인 DB에 저장하게 하는 방식으로 구현하였습니다.



- 그림 메인서버에서의 비동기화 요청

## 플레이룸 영상 싱크

플레이룸 영상 싱크 기능은 앞서 메세징 서비스에 사용했던 Websocket과 STOMP 기술을 활용하여 구현되었습니다. iframe태그로 Embed 된 Youtube 기본 Player에서 제공하는 Player 상태 정보(재생 상태, 비디오 ID, 재생 시점 등)를 바탕으로 일정 주기로 Sync data를 작성하여 STOMP Pub/Sub모델을 이용해 방장이 지속적으로 Publish하는 Polling방식으로 동작합니다.

구상 초기 Youtube Player의 재생 관련 Event에 기반한Interrupt 방식을 고려하였으나 클라이언트의 네트워크 환경, 특히 모바일 네트워크의 불안정성이 크고 기본 플레이어에서 제공되는 Event가 한정적이라는 한계가 있었습니다. 한정적으로 제공되는 Event를 통해 이용자의 행위나 네트워크 등에 따른 Youtube Player의 상태 변화를 정확하게 추적하기 어려웠고, 또한 STOMP Pub/sub 모델에서도 Publish한 정보가 반드시 도착한다는 보장이 되어있지 않다는 문제가 있었습니다. 마지막으로, 방장을 제외한 이용자의 Player 임의 조작을 막을 수는 있으나 그럴 경우 UX상 큰 불편을 초래할 것으로 예상되는 점 등 다방면에서 고려한 결과, 네트워크에 다소 부하가 발생하더라도 Polling 방식을 채택하는 것이 적합하다는 결론을 내렸습니다.

구현 도중, 웹페이지 특성상 동일한 페이지(동일한 플레이룸)에 방장(Sync Publisher)이 다른 클라이언트 혹은 동일 클라이언트로 중복 접근한 경우, 영상 싱크 정보가 중복되어 서로 다른 두개 이상의 Sync 정보가 사용자에게 전달되는 싱크 중복이라는 동시성 문제가 발생하였습니다. 이를 제어하기 위하여 플레이룸 접속 시와 퇴장 시 입퇴장 로직을 구현하였습니다. 이를 통해 접근 시 인가(Authorization) 로직을 구현하여 중복 접근을 확인하고 중복된 클라이언트 모두와 연결을 종료하고 재접속하도록 하는 로직을 통해 동시성 문제를 해결하였습니다.

## Youtube API

플레이리스트 생성하기 및 검색 페이지에서 유튜브 영상을 검색하거나 취향 탐색 페이지에서 카테고리 별 유튜브 영상을 조회할 때 유튜브 API를 사용하였습니다.

영상 검색의 경우, Search API를 통해 기본적인 정보(제목, 썸네일, 채널명)를 요청하였고 추가적으로 필요한 정보(카테고리 ID, 영상 길이)를 Videos API를 통해 요청하였습니다. 이때, 추가적으로 Videos API에 요청할 때 영상 개별로 요청하지 않고 검색한 영상 ID들만 따로 저장하여 한 번의 요청으로 처리하였습니다. 또한, 요청하는 part와 fields를 필요한 부분만 요청하여 요청 데이터를 최소화하였습니다. 기본 데이터 요청과 Quota를 절약한 데이터 요청의 Quota 차이는 다음과 같습니다.

**<기본 추가 데이터 요청(50개 기준)>**

\*50개의 영상을 개별로 요청하며, 모든 part와 fields를 가져오는 경우

Videos - list : 50 \* 21 = **1050**

**<Quota 절약한 검색 추가 데이터 요청(50개 기준)>**

\*50개의 영상을 묶어서 한번에 요청하며, 필요한 part와 fields만 가져오는 경우

Videos - list(snippet/contentDetails) : 1 \* 5 = **5**

취향 탐색 페이지에서 카테고리 별 유튜브 영상을 조회하는 경우, Videos API를 통해 12가지 카테고리 중 해당하는 카테고리 ID의 가장 인기있는 영상들을 요청하였습니다. 이때, 사용성과 유튜브 Quota 절약을 위해 해당 페이지에 방문하여 이미 조회한 카테고리의 경우 Vuex Store에 저장하였습니다. 사용자는 해당 페이지에 머무르면서 카테고리 필터를 여러 번 누르더라도 한번 조회한 카테고리의 경우 API를 또다시 요청하지 않고 Vuex Store에서 가져오기 때문에 빠른 속도로 조회할 수 있습니다. 또한, 매번 유튜브 API 요청 보내는 것을 방지하기 때문에 유튜브 Quota도 함께 절약하였습니다.

## UI/UX

사용자의 UI/UX 를 향상시키기 위해 사용자 행동 최소화, 세부적인 기능 추가, 모달/로딩바/알림창 활용, 유효성 여부 표시 등을 하였습니다. 세부적으로 고려한 UI/UX는 다음과 같습니다.

**[사용자 행동 최소화]**

1) 플레이리스트로 플레이룸 생성 기능

텍스트이(가) 표시된 사진

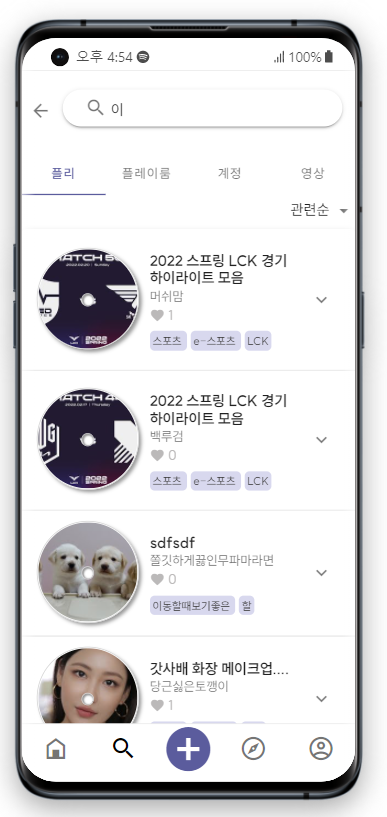
자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

플레이리스트를 기반으로 플레이룸을 생성하는 기능을 제공하여, 플레이룸을 생성할 때 작성해야 하는 노력을 최소화하였습니다. 해당 기능은 플레이리스트를 생성하는 시점과 플레이리스트 디테일 페이지에서 제공하며, 플레이리스트의 제목, 소개글, 태그, 플레이리스트가 플레이룸 생성하기 페이지에 들어간 상태로 작성할 수 있도록 제공합니다.

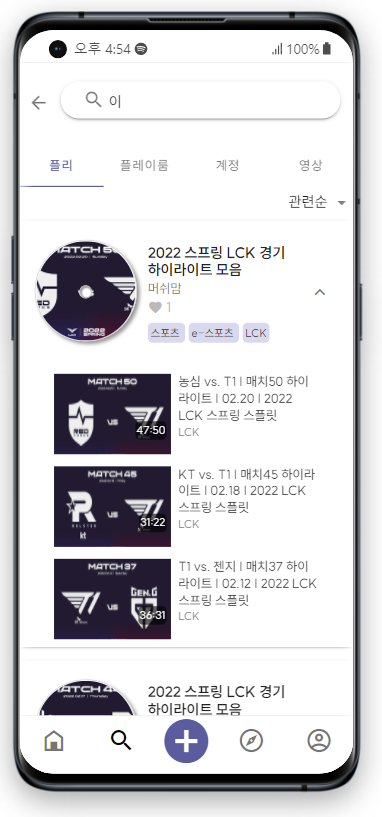
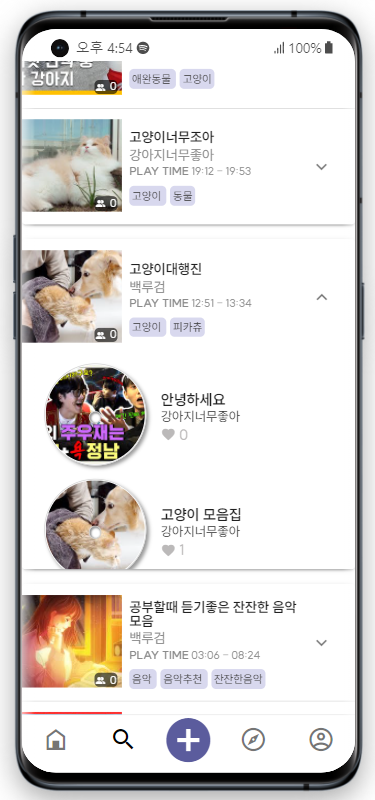
2) 검색페이지에서 탭 이동을 통한 검색 기능

 텍스트이(가) 표시된 사진

자동 생성된 설명 

한 번 검색한 키워드는 매 종류(플레이리스트, 플레이룸, 계정, 영상) 별로 또다시 검색할 필요 없이 탭이동만으로 검색이 가능합니다.

3) 플레이리스트 내 영상, 플레이룸 내 플레이리스트 조회

디테일 페이지로 이동하지 않더라도 화살표 클릭만으로 해당 플레이리스트, 플레이룸 각각에 어떤 영상과 플레이리스트가 있는지 확인이 가능합니다.

4) 비로그인 유저 기능 제공

비로그인 유저에게도 최소한의 기능을 제공하여, 회원가입이나 로그인 없이도 서비스를 이용할 수 있도록 하였습니다.

- 비로그인 유저 제공 기능 : 메인 피드 조회, 검색 페이지 조회, 플레이룸 시청, 취향 탐색 페이지 조회, 타 유저 프로필 조회

**[세부적인 기능]**

1) 설정( 약관, 비밀번호 변경 등)

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트, 전자기기, 영수증, 스크린샷이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

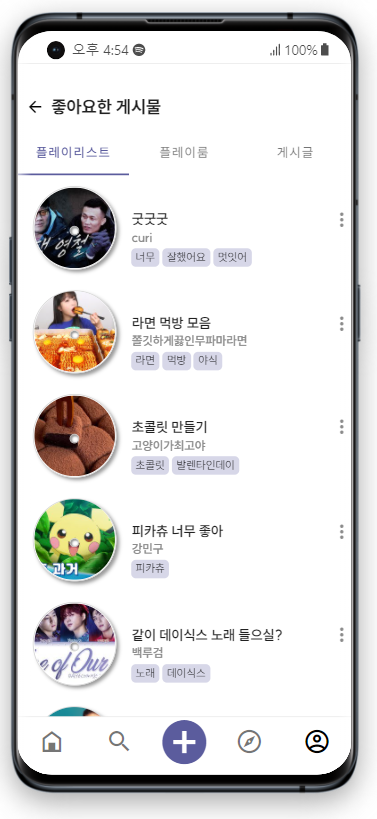
자동 생성된 설명

- 개인정보 이용약관 : TUPLI 서비스 내에서 개인정보 이용에 대한 약관 내용을 제공하여 민감한 정보 이용에 대한 안내를 하였습니다.

- 서비스 이용 약관 : TUPLI 서비스 내 이용에 대한 약관 내용을 제공하여 기본적인 이용 안내를 하였습니다.

- 비밀번호 변경 : 비밀번호를 변경할 수 있도록 제공하였습니다.

2) 좋아요해둔 게시물(플레이리스트, 플레이룸, 게시글) 조회



좋아요한 게시물들을 조회할 수 있도록하여 이를 기반으로 유저의 활동기록을 조회하고, 플레이리스트 생성/플레이룸 생성 등에 도움을 주도록 하였습니다.

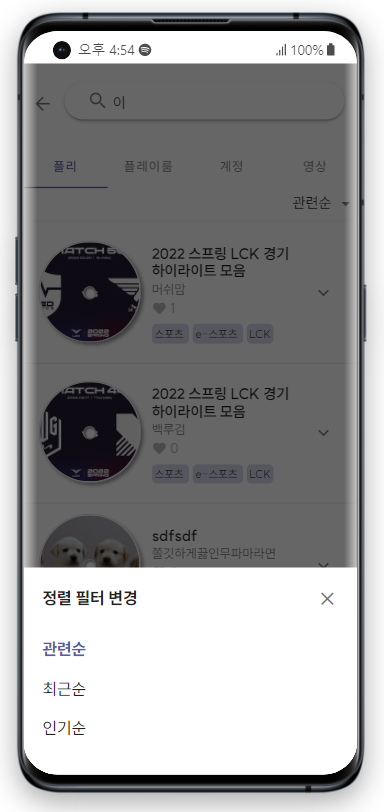
3) 초대/알림 설정

텍스트, 전자기기, 스크린샷이(가) 표시된 사진

자동 생성된 설명

실시간 알림, 초대에 대해 차단하거나 범위를 설정할 수 있도록하여, 유저 별로 각자 원하는 SNS 환경을 조성할 수 있도록 하였습니다.

4) 검색 정렬 기능



검색을 통해 빠르게 정보를 찾을 수 있도록 플레이리스트/플레이룸/계정/영상 각각의 게시물 특징에 맞춰 정렬 기능을 제공하였습니다.

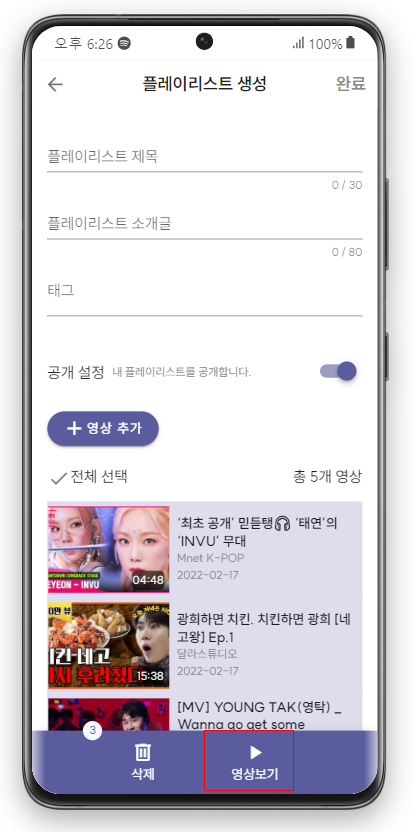
- 플레이리스트 : 관련순/최근순/인기순

- 플레이룸 : 관련순/최근순/시청자순

- 계정 : 관련순/팔로워순

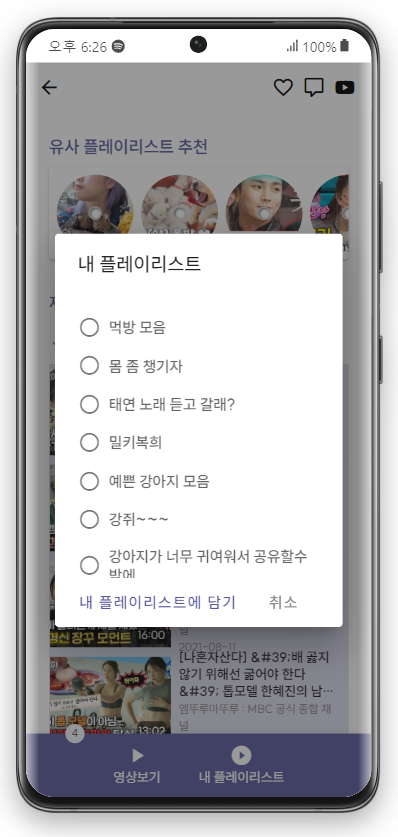
- 영상 : 관련순/최근순/조회순

5) 영상 보기 기능



영상을 추가하거나 탐색할 때 언제라도 어떤 영상인지 확인할 수 있도록 영상들을 묶어서 볼 수 있는 기능을 제공하였습니다. 사용자는 플레이리스트를 생성하며 어떤 영상들을 넣을지 고민할 때, 제대로 영상을 넣었는지 확인할 때 등의 상황에서 영상을 확인할 수 있습니다. 또한 플레이리스트 디테일 페이지에서도 어떤 영상들이 들어있는지 묶어서 바로 확인이 가능합니다.

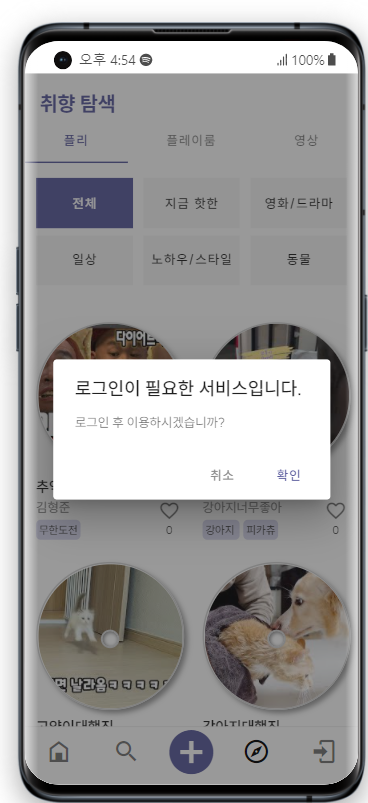
6) 내 플레이리스트에 영상 담기



플레이리스트를 탐색하며 언제라도 마음에 드는 영상을 발견한다면 내 플레이리스트에 담을 수 있는 기능을 제공하여 다른 사람의 취향을 보며 내 취향의 플레이리스트를 구성하는데 편의성을 더했습니다.

**[모달/로딩바/알림창 활용]**

1) 비로그인 유저의 로그인 기능 요청시 모달창을 통한 안내

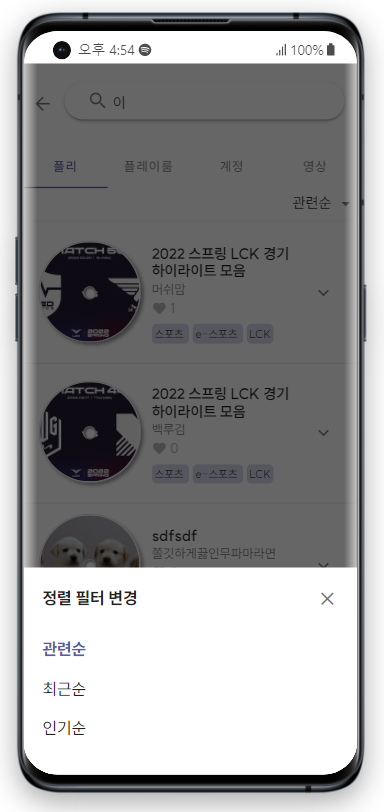


비로그인 유저가 로그인이 필요한 기능을 요청했을 때, 로그인 후 이용할 지 여부를 확인하는 모달창 표시하여 편의성을 높임

2) 요청을 기다리는 동안 사용성을 높이기 위해 로딩바 활용

플레이리스트 생성, 플레이룸 방장 전환, 무한스크롤 등 요청 결과가 올 때까지 기다리는 동안 로딩바를 표시하여 사이트가 멈춘것처럼 보이지 않도록 함

3) 하단 모달을 활용해 모바일 유저의 사용성을 높임



모바일 특성상 하단에 위치한 화면을 클릭하기 쉽기 때문에 정렬/수정/삭제 등의 요청을 할 때 하단 모달창을 활용

4) 유저 행동에 따른 결과 표시

텍스트이(가) 표시된 사진

자동 생성된 설명

유저 행동 요청이 잘 갔는지 확인할 수 있도록 플레이리스트를 생성하며 영상을 추가하거나 삭제할 때 결과를 표시했습니다.

5) 팔로우 요청 실시간 알림

유저의 서비스 사용에 방해되지 않는 선에서 유저 간의 활동을 쉽게 확인할 수 있도록 실시간 알림창을 하단에 구성하였습니다.

## 태그 추천

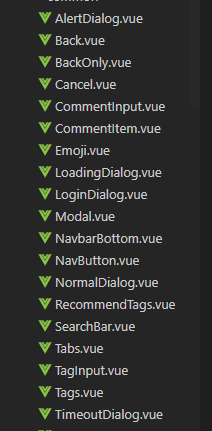
텍스트, 전자기기, 스크린샷이(가) 표시된 사진

자동 생성된 설명

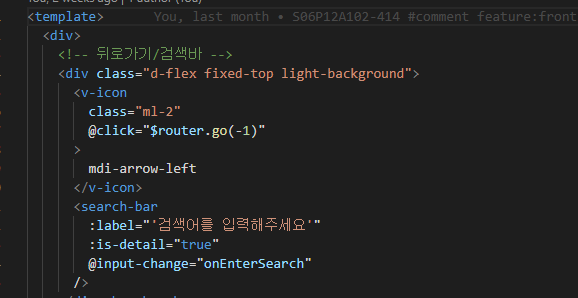
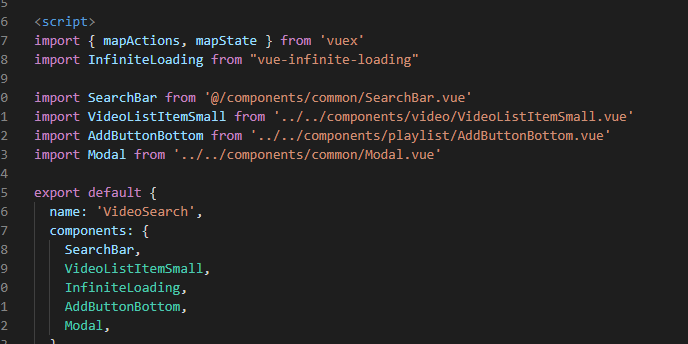
플레이리스트/플레이룸을 생성할 때 유저들이 각각 어떤 내용이 들어있는지 쉽게 살필 수 있도록 태그를 작성할 수 있도록 하였습니다. 이때, 태그를 일일이 입력하려면 유저가 플레이리스트/플레이룸을 생성할 때 번거롭다고 느낄 수 있기 때문에 자주 쓰는 단어들을 인스타그램 트렌드 키워드와 유튜브 트렌드 키워드 중에 3000개를 선정하여 추천하였습니다. 한 글자씩 입력할 때마다 해당하는 키워드를 포함하는 태그를 최대 10개까지 추천하여 보여주도록 하였습니다. 추천한 태그를 클릭하면 입력한 태그 목록에 삽입됩니다.

## FE 컴포넌트/모듈 구성

**[컴포넌트 구성]**

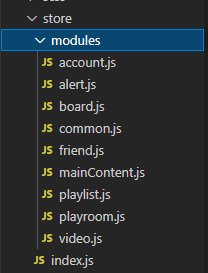
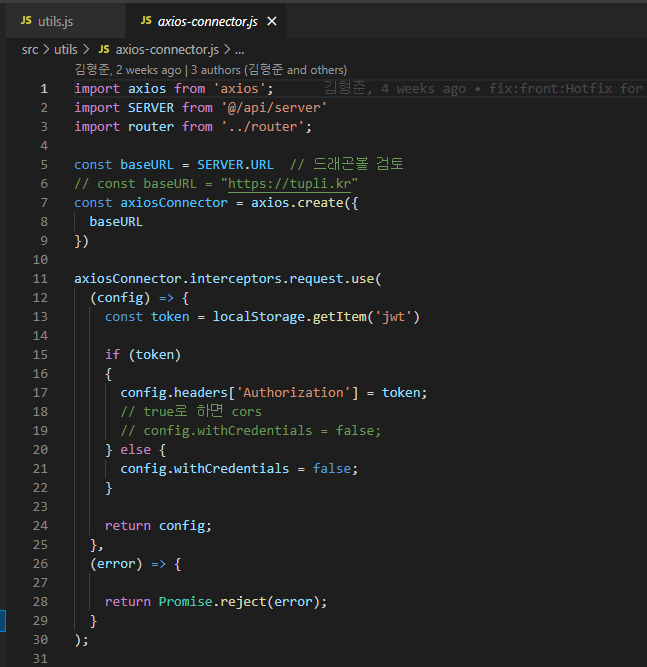
**[컴포넌트 사용]**

검색창, 태그 입력창, 모달창, 뒤로가기 버튼 등 작은 단위로 컴포넌트화하여 통일성과 코드 생산성을 높였습니다. Props를 통해서 해당 컴포넌트가 어떤 용도로 사용하는지 구분하고 template과 script를 커스텀할 수 있도록 하였습니다. 예를 들어, 검색창 같은 경우에는 검색 페이지 뿐만 아니라 플레이리스트/플레이룸/게시글 생성할 때에도 사용되는데 용도에 따라 label 이름을 커스텀할 수 있고 검색용이 아니라 검색 페이지로 이동하는 용도로 사용할 경우 routerPage에 값을 props해서 이동할 페이지를 넣을 수 있습니다.

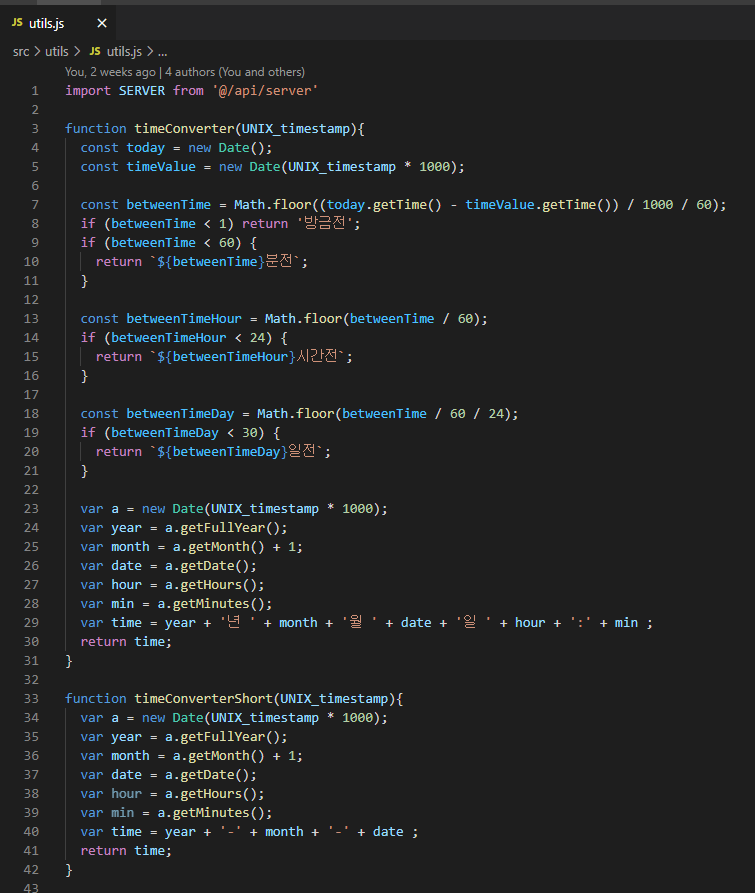
작은 단위로 컴포넌트화하였기 때문에 일일이 코드를 복사해와서 쓸 필요가 없으며 용도 또한 명확하게 알 수 있기 때문에 팀원 간에 코드를 손쉽게 공유할 수 있었습니다. 따라서 두번 작업을 할 필요 없었으며, 같은 용도로 사용하는 컴포넌트는 똑같은 사용성을 제공하기 때문에 사용자에게도 편안한 환경을 제공하였습니다. 또한, 수정사항이 생겼을 경우에도 하나의 컴포넌트만 수정하면 되기 때문에 유지보수를 편리하게 할 수 있도록 하였습니다.

**[vuex 모듈화/axiosConnector]**

컴포넌트와 페이지 별로 구분해서 Vuex Store를 구성할 수 있도록 모듈화하여 분업을 돕고 코드 생산성을 높였습니다. 이때, 서버에 요청을 보내는 axios의 경우 axiosConnector를 따로 생성하여 어느 모듈 안에서라도 갱신된 jwt 토큰을 포함하여 요청할 수 있도록 하였습니다.

**[함수 모듈화]**



자주 사용하는 시간 변환 함수, 프로필 이미지 가져오기, 유튜브 영상 길이 변환 함수 등은 utils.js 파일에 정리하여 export 하였습니다. 이를 통해, 어느 곳에서든 import하여 간단하게 함수를 사용할 수 있도록 하였으며, 변동 사항이 있을 경우 공통적으로 손쉽게 적용할 수 있었습니다.