<PROJECT 중간보고서>

공간에 기반한 음악을 소개하는
웹 서비스 및 API 만들기
My Town Music

TEAM D

유가온, 김다윤

분산시스템 연구실 엄현상 교수님

이대범님(NAVER CORP. NAVER MUSIC)

Table of Contents

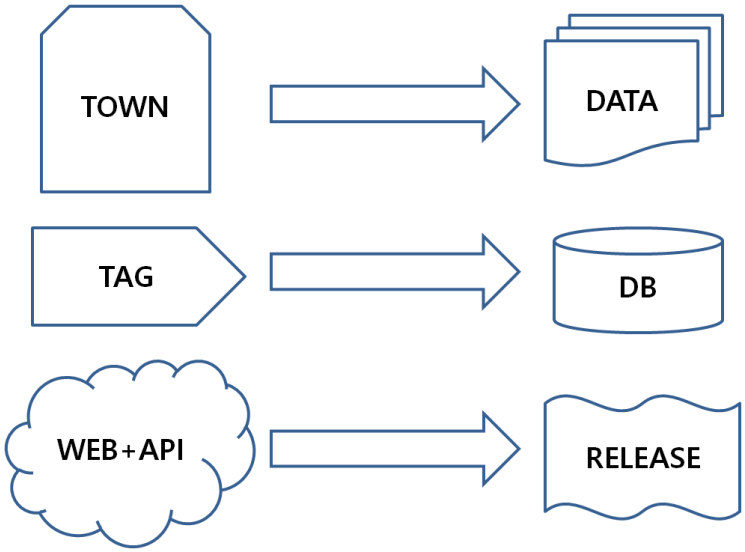
1. Abstrack

구글, 애플, 국내로는 멜론, 벅스를 앞세워 많은 음악 소개 서비스가 만들어져 오고 있다. 하지만 많은 사람들의 삶의 무대가 되어주는 '동네'에 초점을 맞춘 서비스는 없다. 그렇기에 지금까지의 알고리즘들은 장소에 대한 맥락과는 다소 일치되지 않는 양상을 보인다. 따라서 이를 개선하기 위하여 동네의 분위기를 태그와 같은 객관적인 데이터로 구체화하고 음악 데이터와 연계하는 알고리즘이 필요할 것이다. 또한 이 알고리즘이 실행될 어플리케이션을 만들기 오픈 API와 웹 기술을 활용하여 유저친화적인 프론트엔드 페이지와 데이터베이스 지식, restful을 고려하여 백엔드 데이터베이스를 개발하여야 한다. 프로젝트의 마지막 단계에서는 사용자로부터 지도상의 위치를 입력 받아 태그와 함께 음악을 소개해주는 서비스(MTM이라 칭함)의 데모를 시연한다.

2. Introduction

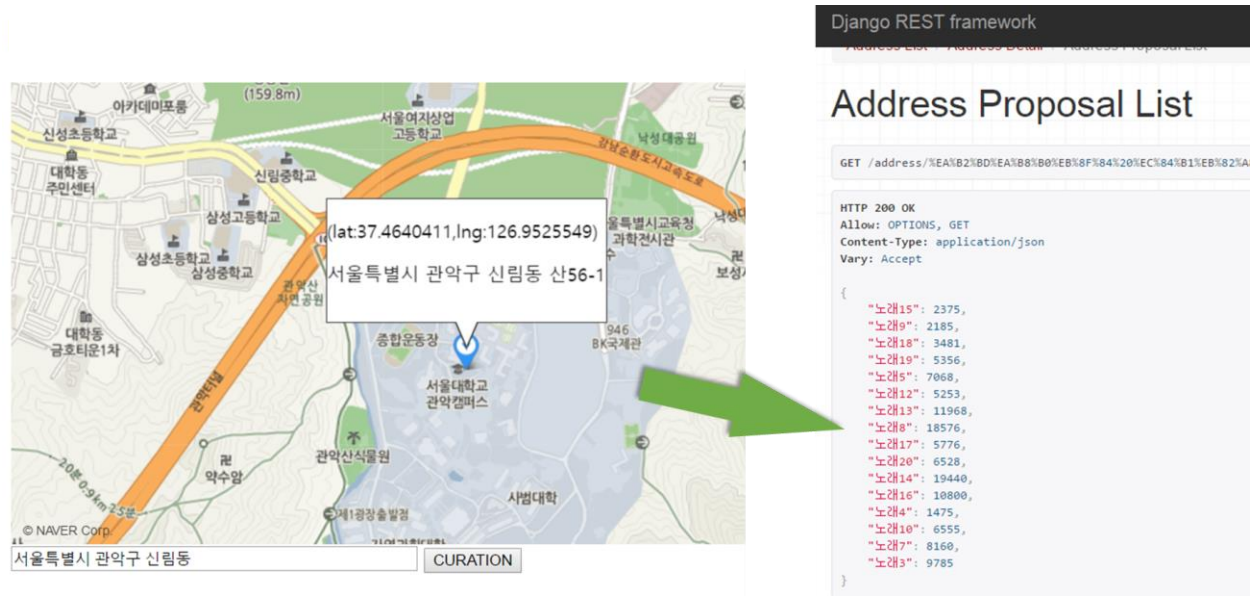
MTM은 장소, 그 중에서도 동네에 기반한 카테고리별 음악 소개 웹 서비스와 API이다. 이러한 시도는 먼저 데이터 측면에서 상당히 유의하다. 동네는 사람들의 주요 활동 범위이므로 양적인 부분과 다양성을 동시에 충족하는 데이터를 얻을 수 있다. 또한 동네마다 지리적, 문화적 특성으로 인하여 분위기가 다르며 국지적인 특성을 가지고 있다. 이로서 상황과 환경에 크게 좌우되는 음악의 성질상 통일성 있는 데이터를 모을 수 있다. 추가적으로 위도 경도 기준으로 기록된 음원 재생 자료를 오차 없이 활용할 수 있어 정보수집 역시 쉽다. 둘째로 데이터 베이스적 측면이다. 어떤 장소와 음악은 태그를 이용하여 자연스럽게 묶인다. 예를 들면, 신나는 장소와 신나는 음악은 매끄러운 조합이라고 할 수 있다. 그러므로 태그를 이용하였을 때 장소와 음악의 관계형 데이터 모델을 만들기 용이하다. 마지막으로 배포와 재배포의 측면이다. 이러한 접근으로 탄생한 응용 프로그램은 웹 형태로 배포되어 인터넷이 가능한 어느 곳에서나 여러 브라우저를 통해 만날 수 있다. 이와 같은 온라인 방식의 배포는 더 높은 접근성으로 이어지며 어플리케이션 이용률

증가, 그리고 많은 리뷰와 피드백으로 이어지길 기대할 수 있다. 나아가 이를 API 형태의 열린 서비스로도 배포하면 2차 앱 개발, 3차 앱 개발로도 발전할 수 있다.



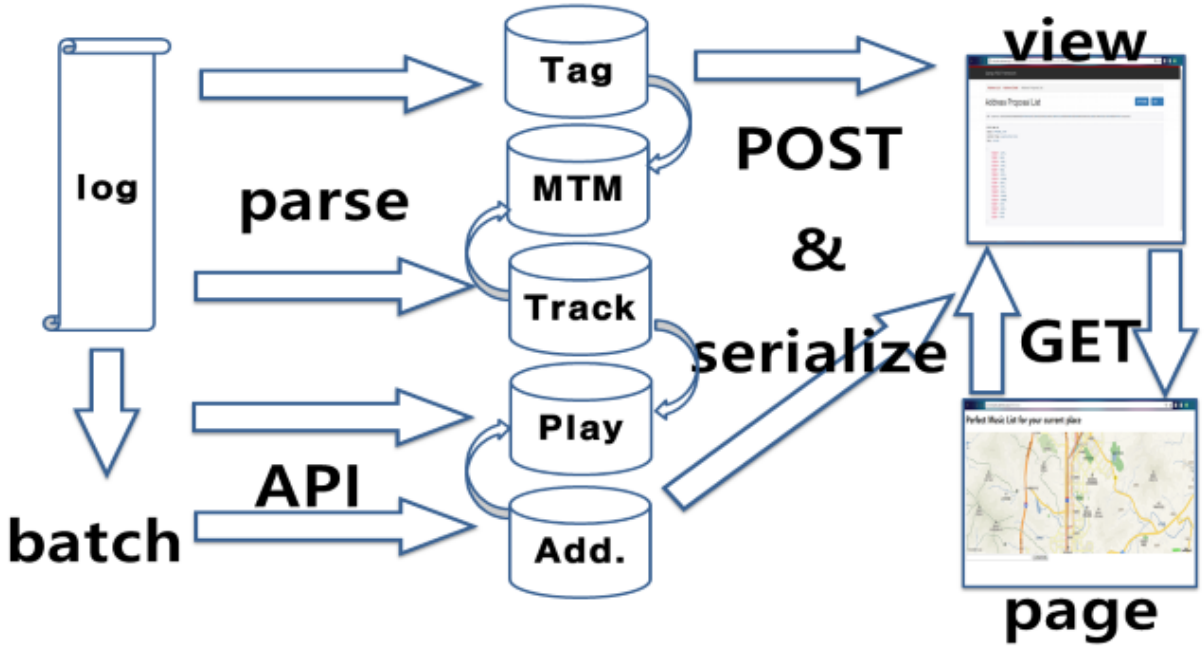
(Figure 1. Project schema)

이를 위한 작업은 크게 앞단과 뒷단으로 나눌 수 있다. 앞단 쪽은 restful, 유저 친화성을 충족하는 웹 서비스와 API를 개발하는 것이다. 다시 웹 서비스는 해당 웹 페이지에 접속하여 지도상에 위치를 눌렀을 때 그 동네의 분위기를 묘사하는 해시 태그와 그에 속하는 음악들을 아티스트와 앨범 단위로 소개해주는 것이다. 그리고 API는 이것을 다른 프로그램에서도 이용할 수 있도록 간단한 프로그래밍 언어의 형식의 인터페이스를 제공할 것이다. 뒷단에서는 안정성 있고 앞단과의 상호작용에 적합한 데이터 베이스 구축과 빠르고 효과적인 알고리즘을 만드는 것이 목표이다. 데이터 베이스는 rest에 맞추어 GET, POST 같은 동작을 수행할 수 있으며 각 데이터를 대표할 수 있는 적합한 model을 설정해야 한다. 그리고 각 모델간 하나 또는 여러 관계를 변환해주는 serializer와 각 변환 결과를 보여줄 url과 view의 폼을 만드는 것이 중요하다. 마지막으로 알고리즘 측면에서는 DB에 의해 정규화된 데이터를 사용하여 일관성 있고 기대에 맞는 결과를 낼 수 있어야 한다. 정확한 이론적인 뒷받침이 있으면 좋겠지만 실용적인 면에서 합리적이고 직관과 맞는 휴리스틱 방법으로 개발한다.



(Figure2. front page IO)

전체적인 데이터의 흐름은 다음과 같다. 회사로부터 지원받은 로그 파일을 파싱하여 http call 중의 POST의 형태로 데이터 베이스에 넣어준다. 그 중 재생 목록 같이 방대한 양의 파일은 람다 아키텍처 모델을 채용하여 배치 모듈을 두어 주기적으로 시간 순으로 오래된 데이터를 삭제하고 새로운 데이터로 갱신하여준다. 또한 위도 경도 형식으로 저장되어있는 위치정보를 API를 이용하여 동 단위로 변환하여야 한다. 이후 데이터를 serialize하고 비로소 view에서 앞단에서 사용가능한 형태로 올바르게 보여지게 된다. 한편 앞단에서는 사용자로부터 지도의 형식으로 위도 경도 단위의 주소를 입력 받게 된다. 이를 뒷단과 마찬가지로 API를 이용하여 동 단위의 주소로 바꾸어 최종적으로 뒷단에서 제공될 restful URL에 primary key로 접근하게 된다. 본래 자바스크립트에서는 지원하지 않지만 Django의 Template 기능으로 url에 http call GET을 보내면 view를 통하여 추천 음악의 리스트를 볼 수 있다. 다시 앞단으로 리스트를 끌어와 추가로 GET을 요청하여 앨범과 아티스트의 정보도 확인할 수 있다. 이렇게 얻은 리스트와 기타 정보들을 미리 디자인 된 패턴에 따라서 한눈에 보기 좋은 방식으로 출력한다. 마지막으로 사용자는 이 정보들을 웹 페이지에서 확인해 볼 수 있다.



(Figure3. data flow model)

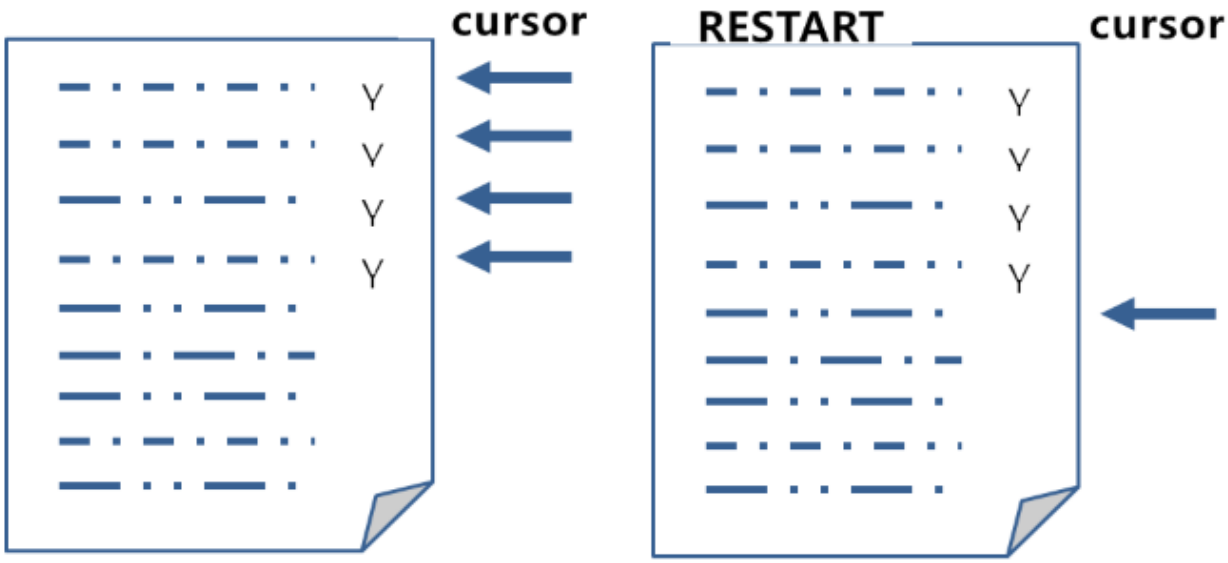
3. Background Study

API (Application Programming Interface)

어떤 응용 프로그램을 다른 프로그램에서 사용할 수 있도록 인터페이스를 제공하는 것이다. 여러 가지 형태로 존재하지만 네이버 지도 API가 자바 스크립트를 이용하는 것처럼 타 프로그래밍 언어와 접목하는 것을 목표로 두었다. 본 프로젝트에서는 Django에서 API를 만드는 툴을 제공함으로 이를 이용한다.

장애 극복(Fail over)

시스템이 알 수 없는 이유로 장애가 생겼을 때 대처하는 방식이다. 이번 프로젝트에서는 아키택쳐 혹은 데이터 특성상 반드시 필요한 배치 모듈이 fail 되었을 때 자동으로 올바른 상태로 돌아가서 작업을 할 수 있도록 한다. 로그 파일의 사본을 두고 그 위에서 작업을 하면서 읽은 표시를 하는 형식으로 진행한다. 한편 쉘에서 프로그램을 재배치하는 일은 Crontab에서 지원한다.



(Figure 4. failover handling)

Generic programming

데이터 형식에 관계 없이 같은 방식으로 프로그래밍 할 수 있는 기술이다. 재사용성이 높아 재사용성이 코드가 간결해지고 오류를 줄일 수 있는 장점을 가진다. 현재 백엔드에서 static generic view를 만들 때 사용한 방식이다.

휴리스틱 알고리즘(Heuristic alogrihm)

```
play[] = playlist.filter(address)          // 위치로 재생목록 선정
for p in play[]:                           // play 목록 중 하나의 play
        for tag[] in p:                    // play와 연결된 tag
                tag[] += ph * ts           // 조화도 * 신뢰도 = tv
for t in tag[]:                            // tag 목록 중 하나의 tag
        for proposal[] in t:               // tag와 연결된 play
                proposal[] += ph * tv // 신뢰도 * 조화도 * 신뢰도
return proposal[]
```

(figure 5. hueristic alogorithm for mtm scoring)

합리성과 직관에 의존한 알고리즘으로 항상 최적의 답을 찾아낼 수는 없지만 생산성과 실용적 측면에서 이득을 얻는 방법이다. NP문제의 근사적인 해법을 빠르게 낼 때에도 많이 사용되며, 느린 알고리즘의 속도를 개선하는 곳에도 사용된다. 그렇기 때문에 데모에서는 Figure 5. 와 같은 효율적인 수식을 사용한다. 먼저 태그 자체의 객관적인 신뢰성이 곱해져 신뢰성이 낮은 태그를 배제할 수 있으며, 플레이 리스트로 태그를 찾을 때나 태그리스트로 플레이를 찾을 때 모두 둘의 조화도를 모두 고려하여 수식을 만들었다. 따라서 한쪽이 태그와 관계가 적으면 반영이 적거나 아예 되지 않는 구조이다. 데이터의 양에 따라서 상위 플레이 목록이나 상위 태그 목록을 얻어 개선 가능성이 존재한다.

Geocoding

일상생활의 주소를 이용하여 위도 경도 같은 절대 좌표를 얻어내는 것을 의미한다. 반대 개념으로 Reverse geocoding은 위도 경도로 좌표로 일반적인 주소를 얻어내는 것을 의미한다. 여러 API에서 지원하고 있으며 NAVER MAP API를 사용하였다.

횡적 확장 구조(Horizontal scailing architecture)

하나의 작업을 여러 서버에서 나누어서 할 수 있는 병렬적인 구조를 의미한다. 로그 파일 처리의 병렬 작업이 가능한 것과 POST 같은 protocol 을 한꺼번에 처리할 수 있으므로 충족한다.

람다 아키텍처(Lambda architecture)

빅데이터와 같이 정보량이 지나치게 많을 때 필요한 정보 기준으로 정보를 갱신해주는 배치모듈을 두는 구조이다. 상대적으로 적은 정보를 사용하여 효율성이 높으며 신뢰성 있는 결과를 보장해준다. 이때 필요한 배치모듈의 스케줄링은 Crontab에서 지원해준다.

관계형 데이터베이스 모델(Relataion database model)

테이블형태로 만들어진 Key와 Value의 관계를 표현해주는 모델이다. 아직까지 표준적으로 사용되는 신뢰성 높은 모델이며 정규화랑도 관련이 깊다. Django에서 SQLite를 기본적으로 지원하여 구현에는 문제가 없다. 이를 심화하여 모델간에 더 넓은 관계인 1:N의 관계나 N:N의 관계를 나타내기 위하여 Foreignkey와 Many to many 모델을 사용하였다. 전자는 Django에서 지원해주고 후자는 Pair의 형태로 직접 구현하였다.

REST(Representational State Transfer)

다음과 같은 6가지를 충족하는 소프트웨어 구조를 말한다. 클라이언트와 서버 구조를 가지고 있으며, 중간 상태를 거치지 않고 결과가 나와야 하며, 캐싱 기능과 계층화가 이루어져있고, 프론트로 확장이 가능하고, 구조가 일관성 있고 단순하여야 한다. Primary key로 접근할 수 있는 Rest URL과 유저와 서버를 통해 소통하며 나머지의 기능은 Django REST Framework에서 지원하며 그렇지 않은 기능은 파이썬으로 구현하였다.

스크립트(Script)

절차를 자동화하여 프로그래머 대신 결과를 내어주는 파일이다. 로그파일을 파싱하여 DB에 넣을 때와 배치서버를 만들 때 Shell script, Python script를 쓴다.

Template tags

파이썬 코드를 HTML로 바꾸어 브라우저가 활용할 수 있게 하는 부분이다. Django template tags로 동적인 웹 페이지를 만들 수 있다. 프론트 페이지를 만들 때 쓴다.

4. Goal/Problem & Requirements

프로젝트의 핵심을 요약하면 크게 4가지로 나눌 수 있다.

첫째로는 서비스 관점이다. 서비스를 구현하는데 필요한 데이터의 스펙을 미리 정의하고 그에 맞는 데이터를 합리적으로 조합하고 이용하여 사용자의 유인요소를 만들 수 있어야 한다. 또한 실제로 수 차례 이용해보면서 쌓인 경험을 반영한 데모를 만들 수 있어야 한다.

두 번째로는 데이터 베이스의 활용이다. 서비스 정보를 적절하게 테이블 스키마로 표현해야 하고 각 데이터에 성질을 잘 나타내어주며 접근 가능한 Key를 부여하고 주제에 적합한 DBMS를 사용할 수 있어야 한다. 또한 쿼리를 작성하여 DB와 원활하게 정보 교환이 일어나도록, 그것을 로그로 남기도록 처리해야 한다. 일련의 과정에서 항상 안정성이 보장되어야 한다. 추가로 병렬처리를 위한 횡적 확장이 고려된 설계를 해야 한다.

세 번째로 로그 수집 및 가공 역시 중요하다. 주기적인 로그 취합과 그것을 DB에 넣기 위해 가공하는 과정, 올바르지 않은 데이터 처리가 모두 되어있어야 한다. 또한 failover가 가능하여야 한다. 실제로 서비스가 가능하고 손쉬운 사용이 가능한 API를 만들 수 있어야 하며 해당 과정에 REST를 적용하여야 한다. 코드의 품질 또한 고려해야 할 대상이다.

마지막으로 테스트 환경을 들 수 있다. 일관성 있는 테스트 스크립트를 작성해야 하고 다양한 테스트를 위한 환경이 있어야 하며 실제 테스트를 통해 병목 지점을 찾고 TPS 측정을 통한 서버 스펙 산정이 가능하여야 한다.

5. Approach

위의 각 항목에 대하여 접근방법은 다음과 같다.

로그 파일을 분석하여 사용할 수 있는 부분의 데이터를 정의하여 스펙을 구성하였다. 내부적으로는 기존과 차별화를 두기 위하여 독특한 개념을 활용한 휴리스틱 알고리즘을 개발하였고, 외부적으로는 Django template tags를 이용하여 동적인 웹 페이지 개발을 하였고 bootstrap을 통해 심미성을 확보할 수 있다. 그리고 차별화된 알고리즘을 사용하여 충분한 유인요소를 만들 수 있다. 이를 실제로 공개된 도메인 상의 웹 페이지에서 구동하여 여러 사람의 피드백을 받을 수 있다.

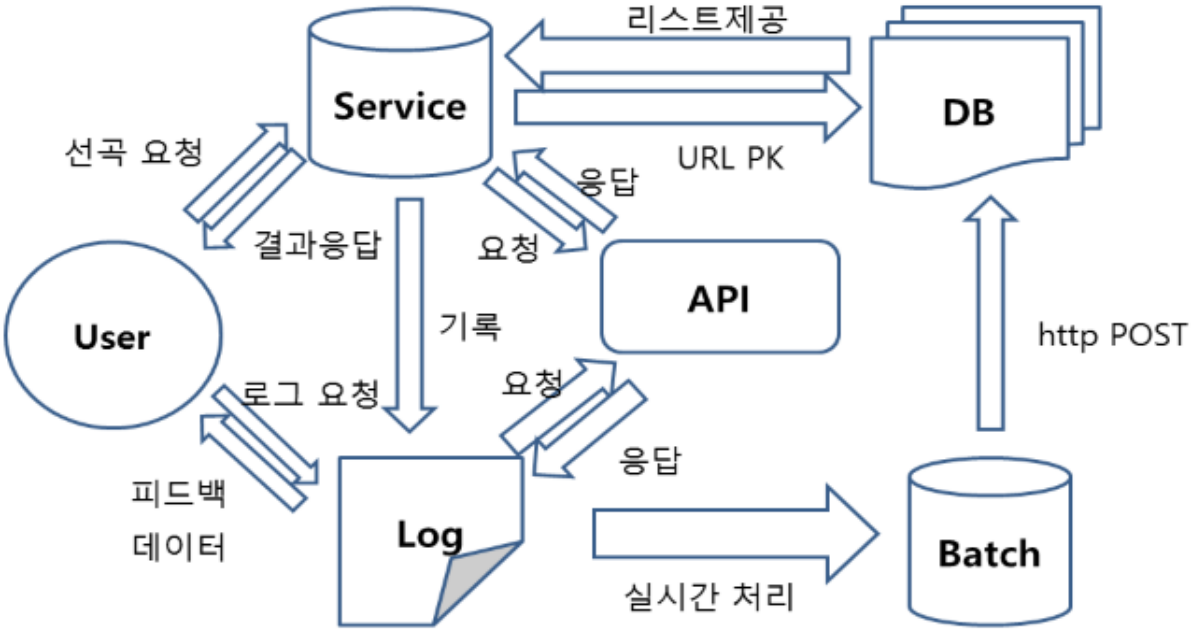
데이터 베이스 관련 접근은 개발 언어로 Django REST Framework를 선정함으로써 크게 이루어졌다. 관계형 데이터베이스 모델 스키마와 모델 선정은 여러 시도를 해보면서 프로젝트에 가장 적합한 것을 얻을 수 있었고 REST URL, Query를 지원받아 보다 안정성 있고 쉬운 정보 교환이 가능하다. 앞선 행적들을 로그로 남기는 것은 훨씬 간단하게 되었다. 이들이 모두 REST 성질을 만족하므로 횡적확장도 충분히 가능할 것으로 보인다.

로그 부문에서는 Primary Key를 Unique ID로서 서로 겹치지 않게 설계하였고 Generic 개념을 도입하여 코드 재사용성을 늘리는 한편 데이터를 핸들링에도 큰 도움이 되었다. 주기적인 데이터 취합은 현재 쉘 스크립트를 사용하고 있으나 최종 데모에서는 Crontab을 사용한 Python 스크립트를 이식할 것이다. Crontab과 로그 표식 메소드로 failover 시스템도 설계할 수 있다. 이로써 많은 것들이 고려된 웹 페이지와 API를 만들 수 있다.

테스트 관련 접근은 계획상 조금 더 후반부에 있지만 파이썬 스크립트로 유사한 백엔드 테스트를 진행한 바 있으며 셀레늄, 제스트 같은 여러 도구들을 물색하고 있다.

6. Project Architecture

MTM의 전체적인 구조는 다음과 같다.



(Figure 6. MTM architecture)

데이터베이스를 기준으로 데이터의 흐름을 나눌 수 있다. 하나는 데이터베이스에 쓰는 것이고, 하나는 읽는 것이다. 쓰는 과정에서는 다른 재생 로그 파일, 트랙 로그 파일들이 배치 모듈을 통해 실시간으로 DB에 맞는 모델로 정규화되고 파싱되어 DB에 쓰이게 된다. 이 과정에서 일부 데이터를 변환하는 과정에 API를 이용하기도 한다. 적절히 처리된 데이터들은 POST을 통해 DB에 전송된다. 마지막에 DB에서는 이것들을 serialize하고 view를 통해 외부에서 접근 가능하게 된다.

읽는 과정에서는 반대로 DB의 view가 있는 URL에서 정보를 가져온다. 프론트엔드에 사용자가 선곡을 요청하면 프론트엔드인 서비스 모듈이 input을 가지고 API에 요청을 하여 DB에 접근할 수 있는 PrimaryKey를 얻는다. 이제 미리 스크립트에 작성된 URL과 PK를 합쳐서 REST URL로 DB에 view에 접근하여 원하는 리스트를 얻을 수 있다. 이것을 다시 서비스 모듈에서 사용자에게 렌더링하여 보여주면 된다. 이외에도 과정으로 서비스는 사용자의 요청 로그를 기록하고 나중에 개발자가 로그 파일을 요청 했을 때 피드백 데이터를 넘겨주는 과정이 있다.

백엔드 디렉터리 구조와 파일은 다음과 같다.

```
music(root)
├──── music - setting.py urls.py
|     └──── __pycache__
└──── propose - models.py serializers.py views.py
|     ├──── migrations - SQLiteDB
|     |     └──── __pycache__
|     ├──── __pycache__
|     └──── templates
|           └──── propose - test.html
└──── script - *.sh *.py *.txt
```

(Figure 7. backend directory)

먼저 music은 Django를 이용해 생성한 프로젝트 디렉터리의 가칭이며, 마찬가지로 propose는 Django로 만든 개발중인 App의 임시적인 이름이다. music은 프로젝트의 생성일, 포함된 어플, 사용하는 Django 정보 같은 개괄적인 정보를 담은 settings.py, 전체 URL이 담긴 urls.py가 있다.

propose에는 모델이 정의된 models.py, serialize 기능을 하는 serializers.py, DB의 접근을 허용해주는 views.py가 있다.

그 하위 폴더인 migrations에는 모델 정보와 SQLite DB 정보가 담겨있으며, 다른 하위 디렉터리인 templates에 페이지 소스가 정의된html 코드가 있다. 한편 이들과 별개로 script가 존재한다. script에 있는 각종 스크립트들은 쉘스크립트의 확장자인 sh나 파이썬의 확장자인 py를 가지는데 같은 목록에 있는 동명의 텍스트 형식 로그 파일을 읽어 POST 요청을 자동으로 해주는 역할을 맡는다.

마지막으로 __pycache__는 프로젝트 진행 중에 자동적으로 생성되는 캐시가 담기는 장소이다.

프론트엔드 URL 구조는 다음과 같다.

r'/test/
r'/track/
 └─── r'/track/<pk>/
r'/tag/
 └─── r'/tag/<pk>/
r'/mtm/
 └─── r'/mtm/<pk>/,
r'/play/
 └─── r'/play/<pk>/
r'/address/
 └─── r'/address/<pk>
     └─── r'/address/<pk>/proposal/

(단, r'은 http://wlxyzlw.iptime.org:[port_num], <pk>는 해당 모델의 primary key)

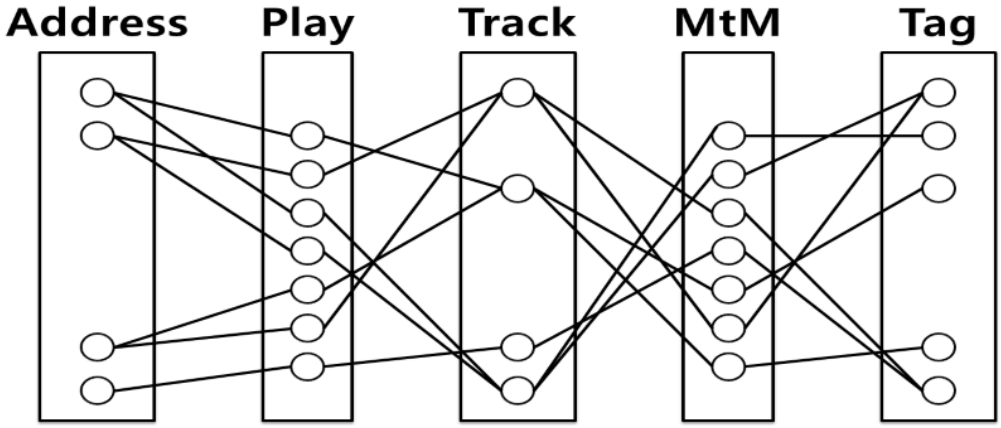
(Figure 8. frontend url)

각 기능은 다음과 같다.

tag 는 태그의 name 과 신뢰도인 score 를담고있고 POST/GET 을 지원.
track 은곡의 id 와 titile, 타이틀곡 여부인 is_title 를 가지고 있고 POST/GET 지원.
address 는 주소와 임의의 id 를 가지고 있고 POST/GET 지원.
play 는 created_date 와 track_id, address 를 가지고 있고 POST/GET 지원.
mtm 은 tag 와 track 을 pair 형태로, 조화도 harmony 를 가지고 있고 POST/GET 지원
또한 각각의 모델의 retrieve 페이지 pk 로 접근 가능하다.
이 중 특수하게 r'/address/<pk>/proposal 는 GET 만을 지원한다. 그 이유는 프론트엔드의 GET 요청이 address 와 함께 들어오면 곡들의 리스트가 들어올 부분이 되기 때문이다. 따라서 POST 를 지원하지 않아도 된다. 다음은 예시이다.

~| http GET http://wlxyzlw.iptime.org:8008/address/서울시%20 관악구/proposal
~| {track : ABC}

백엔드 DB 구조는 다음과 같다.



(Figure 9. DB model dependency architecture)

그리고 각 model의 구성은 다음과 같다.

class Track = track_id(Char), track_title(Char), is_title(Boolean)

class Tag = tag_name(Char), tag_score(Integer)

ordering : -tag_score

class mtm = id, mtm_date(DateTime), pair(track,tag), harmony(Integer)

ordering : -harmony

class Address = address(Char), play_num(Integer)

ordering : -play_num

class Play = id, play_date(DateTimeField), pair(Track,Address)

ordering : -play_date

(단, 가장 앞에 있는 field 가 각 model 의 pk 가 된다.)

(Figure 10. modeling detail)

Track 은 자체 id 와 재생 곡 이름, 타이틀곡 여부가 있다.

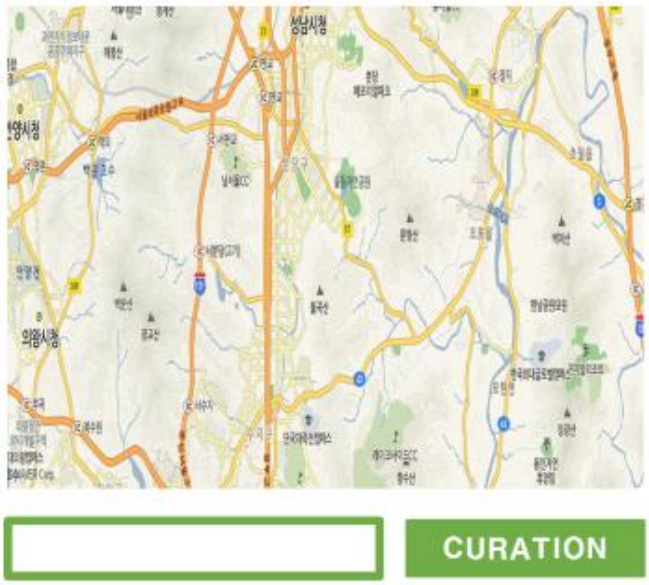
Tag 는 태그 이름과 신뢰도 점수를 가지며 점수가 큰 순서대로 정렬된다.

mtm 은 address 와 play 를 pair 로 가지며 조화도를 추가로 가지고 있다. 때문에 조화도가 큰 순서로 정렬되어있다.

address 는 자체 id 와 지명을 가지고 있으며, 해당 위치의 재생로그 개수가 있다. 재생로그 개수가 큰 순으로 정렬되어있다.

play 는 자체 id 와 재생 시각, 그리고 track 과 address 를 pair 로 가지고 있다. 정렬 순서는 날짜가 최신인 순이다.

프론트엔드 웹페이지 구조는 다음과 같다.



Title	MTM Score	Category
붉은노을	1923	신나는_음악
밤편지	63	혼자_듣기_좋은_노래
Yesterday	723	신나는_음악
아메리카노	16	노동요
오랜날오랜밤	82	감성_터지는_밤

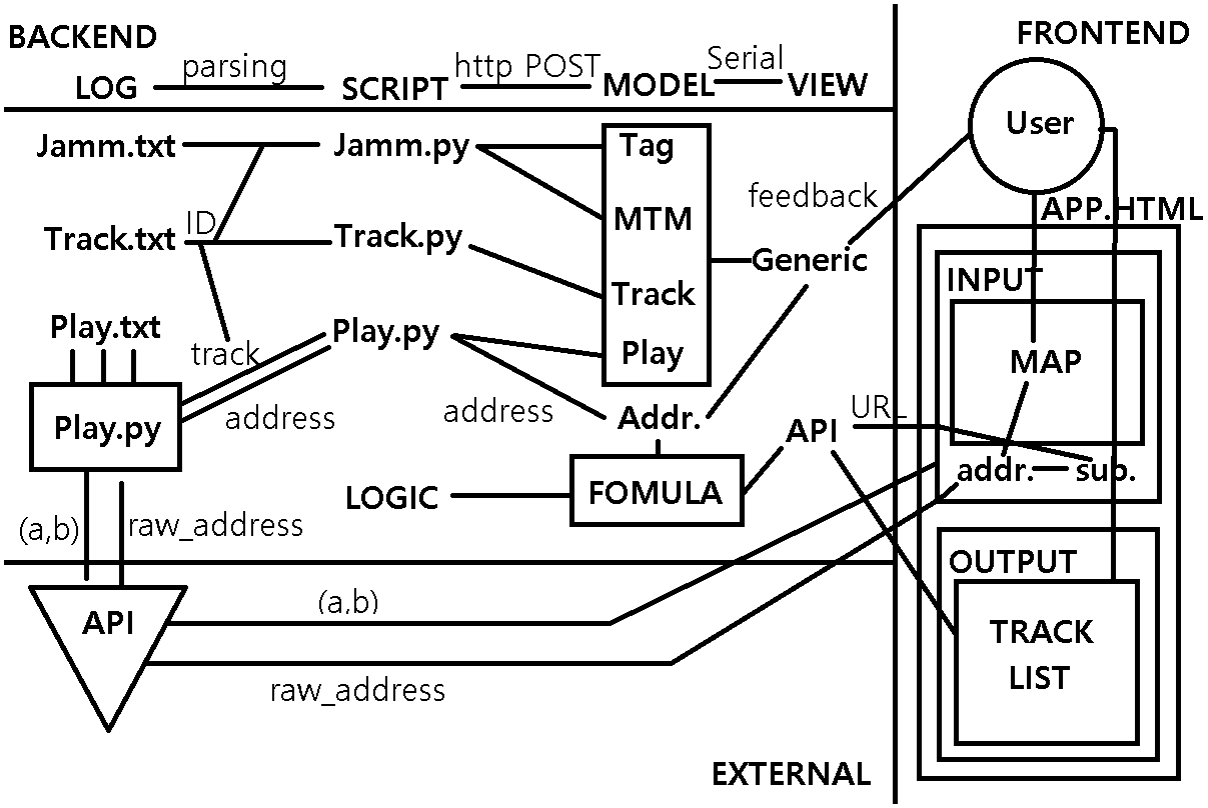
CURATION

(Figure 11. webpage architecture)

프론트 웹 페이지는 4 가지 구성요소가 있다. 각각 API 가 구동되는 지도와 그때의 위치를 말해주는 텍스트 상자, 백엔드에 좌표를 전송하는 버튼, 백엔드로부터 받은 결과를 표시하는 큰 텍스트 상자이다. API 와 서로 정보를 주고 받는 작업은 자바스크립트를 이용하였고 지도를 이용해 유저로부터 입력을 받는 친화적인 환경을 마련하였다. 또한 이 과정에서 위치로부터 좌표를 얻어내는 Geocoding 과 좌표로부터 위치를 얻어내는 Reverse Geocoding 을 둘 다 사용하였다. 전자는 사용자가 자신의 위치를 지도상에서 클릭했을 때의 위치를 위도 경도로 바꾸는 것이고, 후자는 위도 경도 데이터를 주소로 바꾸는 것이다. 양쪽 과정 모두가 API 를 통해 일어난다. 특히 reverse geocoding 과정에서 중간 과정인 전체 주소를 파싱을 해주어야 DB 에서 사용할 수 있으므로 동, 읍, 리 같은 많은 주소 단위를 고려해야 한다. 이렇게 가공된 데이터는 텍스트 상자로 들어가게 되고, 유저는 이를 통해 자신이 의도한 정보와 백엔드로 전송되는 정보가 같은지 직접 판단할 수 있다. 그것이 맞다고 생각되면 전송 버튼을 누르고 백엔드의 URL 에 GET 을 보내서 돌아온 응답을 같은 페이지 상에서 텍스트 상자를 통해서 보면 된다.

7. Implementation Spec

A. Input/Output Interface & Inter Module Communication Interface



(Figure 12. Implementaion spec)

B. Interface Description

스펙은 4 가지-앞단, 뒷단, 외부, 로직- 파트로 나뉠 수 있다. 앞단에서는 User 로부터 App.html 에서 map 를 통해 mouse click 의 형태로 input 을 받는다. 그러면 내부 JS 형태의 script 가 input 를 외부로 넘겨서 위도 경도 coord 의 형태로 변환을 한다. 이 coord 를 다시 외부로 넘겨서 raw_address 로 받아와 파싱까지 완료한 데이터가 address 이다. 이것을 address_box 에서 보여주고 User 는 다시 이를 자신이 생각한 정보와 맞는지 검토한다. User 의 의도와 다르다면 input 을 받는 과정으로 돌아간다. 의도에 모두 부합하면 curation 버튼을 눌러 address 를 pk 로 삼아 뒷단의 propose URL 로 접근하여 View 를 통해 track list 를 가져온다. 이를 rendering 하여 같은 페이지에 track_box 에 띄워주면 된다. 뒷단에서는 DB 를 미리 구성하기 위하여 많은 사전 처리가 필요하다. input 이 되는 log 파일은 3 종류가 있다. jamm.txt 은 tag 의

이름인 name 과 그 tag 에 속해있는 track, 둘 사이의 조화도인 harmony, 태그 자체의 신뢰도인 score, 생성 날짜 date 가 존재한다. track 정보는 track.txt 의 id 에서 참조하게 된다. 이를 Jamm.py 가 Tag 에서 쓰일 ID, name, score, date 와 MTM 에서 쓰일 Track, harmony 로 파싱한다. track.txt 는 pk 인 ID 와 제목인 title 과 타이틀곡 여부인 is_title 정보를 가지고 있다. 여기서 ID 는 jamm.txt, play.txt 에서 참조된다. 이를 Track.py 에서 전부 파싱한다. 마지막인 play.txt 생성된 날짜인 date 와 track.txt 의 ID 를 참조하는 track, 위도와 경도가 포함되어있다. 한편으로 상당히 큰 용량을 가지고 있어 배치 스크립트인 Play.py 를 두어 일정한 시간 간격으로 파일을 나눠서 읽는다. 모듈은 1 차적으로 로그파일을 파싱하여 http GET 으로 외부로 보내는 역할을 맡는다. 이로 얻은 정보인 raw_address 를 파싱하여 address 로 바꾸고 미리 파싱된 데이터와 함께 track, date, address 로 합친다. 각 script 들은 파싱된 정보를 바탕으로 http POST 를 REST URL 에 보내게 된다. Jamm.py 는 Tag 와 MTM 를 구성하게 되고 Track.py 는 Track 을, Play.py 는 Play 를 만들게 된다.

이렇게 만들어진 DB 는 로직에 따라서 serializers.py 에 의해 serialize 되어 REST URL 로 generic view 를 통해 보여지게 된다. 예외적으로 propose 는 휴리스틱 알고리즘에 따라 track 과 추천 점수인 dictionary 인 API view 의 형태로 제공된다. 이로서 앞단에서 http GET 요청이 들어오면 응답을 할 수 있게 된다. 외부를 구성하는 API 는 JS 를 통해 Map 에서 넘어온 정보를 좌표로 바꿔주기도 하고 좌표를 raw_address 로 바꿔주기도 한다. 또한 Play.py 에서 JSON 양식의 http GET 요청을 받기도 한다. 마지막으로 로직 파트는 앞서 휴리스틱 알고리즘이 수식으로 정의되어있고 models, data flow 의 스키마적 형태이기도 하다. 따라서 성능의 개선은 주로 이 부분에서 이루어진다.

8. Current Status

MTM은 지금까지 작업으로도 충분히 제 기능을 하고 있다.

다음은 현재까지 완전히 완료된 사항이다.

1. 작업 환경 구축
2. 횡적 확장을 고려한 백엔드의 개발
3. href를 이용한 임시 프론트 페이지 개발
4. 쉘 스크립트 개발
5. 임시 로그파일 작성
6. 임시 배치 모듈 개발
7. 요청 로그 파일 쉘 로그 형태로 저장
8. 임시 테스트
9. API를 이용해 유저 친화성 고려

9. Future Work

그러나 다음은 더 좋은 기능을 하기 위하여 개선이 필요하다.

다음은 아직 완료되지 않은 사항이다.

1. 추가로 Crontab 설치 필요
2. 백엔드 로직 개선
3. http GET 이 가능한 프론트 페이지로 개선.
4. 쉘 스크립트에서 파이썬 스크립트로 개선
5. 담당 회사로 부터 풍부한 로그 파일 획득
6. Crontab 을 이용한 배치 모듈 개발
7. 요청 로그 파일 txt 로 저장
8. 테스트 스크립트 python 작성
9. API 다른 기능을 추가한 개선

추가적인 과제로 다음과 같은 사항도 있다.

1. failover system 구축
2. MTM 을 API 으로 재개발

10. Division & Assignment of Work

항목	담당자
DB	유가온
웹 페이지	김다윤
설계, 로직	유가온
디자인	김다윤
구조 설계	유가온
테스트 코드	김다윤
서버 관리	유가온

(Figure 13. assignment table)

2명에서 한 팀이어서 역할을 위와 같이 나누었으나 실제로는 대부분의 작업을 같은 시간, 같은 장소에서 함께했다.

11. Schedule

Date	**중간발표**	**5월**		**6월**			**최종발표**
		3	**4**	**1**	**2**	**3**	
작업환경	**O**						**O**
앞단	**O**	**O**					**O**
뒷단	**O**	**O**	**O**				**O**
배치	**O**	**O**	**O**	**O**			**O**
로그		**O**	**O**	**O**			**O**
스크립트		**O**	**O**	**O**			**O**
테스트					**O**		**O**
failover				**O**	**O**		**O**
API				**O**	**O**		**O**

(Figure 14. Schedule table)

12. Demo Plan

시연 계획은 다음과 같이 세 단계로 이루어져있다.

첫 번째는 청중의 프로그램 작동원리에 대한 이해를 돕기 위한 예비 단계이며, 다음과 같은 순으로 진행된다.

데모에서 필요한 예시 로그 파일을 미리 생성해놓는다. 이후 다음과 같은 순서로 예시 스크립트를 돌린다.

track.sh/tag.sh

mtm.sh

address.sh

play.sh

그때마다 각 모델의 view 가 있는 url 에 들어가서 청중에게 확인을 시킨다. 각 DB 모델이 관계를 이루는지에 대한 직관을 줄 수 있다.

그 후 propose 가 있는 url 에 들어가서 예시 결과를 보여준다.

예시) http://wlxyzlw.iptime.org:8008/address/서울시%20 관악구%20 인헌동/propose

이 과정에서 figure 5 의 휴리스틱 알고리즘 원리에 대해 설명할 수 있다.

두 번째는 프로젝트의 최종 산출물인 현실과 가까운 데모를 시연한다. 실제 데이터를 가지고 실시간으로 DB 가 갱신되어 결과가 바뀌는 동적 어플리케이션을 보여주는 것을 목표로 한다. 다음과 같은 진행을 가진다.

먼저 프론트 페이지를 열고 프론트 페이지의 원소들의 각 기능을 설명한다. 필요한 경우 코드의 일부를 공개하여 보다 자세한 부분을 살펴본다. 그리고 직접 설계된 대로 프론트 페이지로 백엔드 페이지에 요청을 보내어 결과를 받아온다. 이 과정은 특별히 청중에게 사용자 친화적인 설계가 반영됐음이 충분히 느껴지도록 그렇지 않은 버전과 대조를 한다. 그리고 데이터 특성에 따라 미리 나눠놓은 결과 케이스들을 함께 실행해보며 합리적임을 납득시킨다. 현 단계를 끝내기 전에 횡적 확장에 대한 고려와 failover 기능이 코드 상에서 잘 마련되었음을 심도 있게 언급하고 질문을 받는다.

마무리 단계로 직접 작성한 테스트 코드를 실행시켜본다. 이 단계에서는 청중에게 마지막으로 프로젝트를 검토하는 과정을 보여줌으로써 각 모듈의 기능에 대한 종합적인 사고를 가능하게 한다. 따라서 각각의 테스트 코드가 모듈 단위까지 독립적인 분담을 하여 각 코드의 역할을 세세하게 알아보도록 작성한다.

[Appendix] Detailed Implementation Spec

프론트엔드 App.html 의 구성 함수

naver.maps.InfoWindow(string 위치)return 지도;

위치의 지도를 띄우는 함수

naver.maps.Event.addListener(double 위도, double 경도, 마우스 위치)return string 동주소;

마우스 클릭으로 위도 경도를 받아 API 호출 후 파싱하여 동주소로 변환하는 함수. 조건문 if(parts[i].slice(-1)=='동'||parts[i].slice(-1)=='읍'|| parts[i].slice(-1)=='면')을 사용해 파싱함.

naver.maps.LatLng(double 위도, double 경도)return API 용 좌표

지도 내부용 좌표 생성 함수

hyper() return 페이지 이동;

미리 계산된 동주소를 이용하여 패이지 이동 함수

http://wlxyzlw.iptime.org:8008/address/"+(값)+"/proposal"로 이동

기타 렌더링 관련 함수들이 존재.

백엔드 Propose 의 구성 파일

Jamm.txt

잼아이디(SHA-1 암호화), 태그명, 태그점수(인기도), 생성일, 등록된 트랙 ID

Track.txt

트랙아이디(SHA-1 암호화), 트랙 타이틀(20 글자 이상은 제거), 타이틀곡 여부

Plat.txt

시간, 재생한 트랙아이디(SHA-1 암호화), 위도, 경도

다음은 각 로그를 각 모델에 POST 해주는 shell script 이다.
address.sh
mtm.sh
play.sh
tag.sh
track.sh

다음은 배치 테스트용 shell script 이다.
test.sh
10 줄을 읽어서 echo 해주고 1 초를 쉬는 배치의 역할을 한다.

serializers.py
TrackSerializer(serializers.ModelSerializer):
트랙 정보를 serialize 하여 트랙 뷰로 ('track_id','track_title','is_title')를 내보냄.
TagSerializer(serializers.ModelSerializer):
태그 정보를 serialize 하여 태그 뷰로 ('tag_name','tag_score')를 내보냄

MTMSerializer(serializers.ModelSerializer):
MTM 정보를 serialize 하여 MTM 뷰로 ('mtm_id','mtm_date','track','tag','harmony')를 내보냄

PlaySerializer(serializers.ModelSerializer):
재생 정보를 serialize 하여 플레이 뷰로 ('id','play_date','track_id','address')를 내보냄
AddressSerializer(serializers.ModelSerializer):
주소 정보를 serialize 하고 추가로 그 위치의 재생 목록 갯수를 get_play_num 로 가져온다. 이후 어드레스 뷰로 ('address','play_num')를 내보냄.

get_play_num (self,obj):
어드레스 필터를 이용하여 재생이 해당 위치에서 되었는지 확인하고 count 함.

ProposalSerializer(serializers.BaseSerializer)`:
to_representation 을 호출한다.
to_representation(self,obj):
figure 5.를 이용하여 mtm score 를 생성하고 트랙의 이름과 함께 리턴한다.

views.py
다음은 각 모델의의 generic view 이다.
List view 와 Retrieve view 가 같이 있음. GET 과 POST 가능.
TrackList(generics.ListCreateAPIView):
TrackDetail(generics.RetrieveAPIView):
TagList(generics.ListCreateAPIView):
TagDetail(generics.RetrieveAPIView):
MTMList(generics.ListCreateAPIView):
MTMDetail(generics.RetrieveAPIView):
PlayList(generics.ListCreateAPIView):
PlayDetail(generics.RetrieveAPIView):
AddressList(generics.ListCreateAPIView):
AddressDetail(generics.RetrieveAPIView):

다음은 proposal list 의 api view 이다.
address_proposal_list(request, pk):
해당 pk 에서의 추천 트랙 리스트를 GET 할 수 있다.

다음은 장고 템플릿 view 이다.
test(request):
테스트 페이지가 정의된 test.html 을 보여준다.

urls.py
백엔드에서 view 를 보여줄 REST URL 이 정의되어 있다.